Web前端开发技术 丛书

零基础 · 项目驱动 · 入门到精通







微信小程序开发实战

微课视频版

1600分钟视频讲解



周文洁 著

超值赠送

- 教学大纲、教学课件、程序源码
- 工具插件、行业案例、视频资源
- 课后拓展作业、期末试卷、在线教程

完整的项目案例： 包括API应用、游戏开发、全栈开发、云开发和第三方UI组件的应用等21个项目案例

全程视频讲解： 1600分钟的视频讲解，真实操作演示，让学习更轻松、快捷

丰富的配套资源： 提供教学大纲、教学课件、程序源码、期末试卷，所有程序均在真机中调试通过

清華大学出版社

Web 前端开发技术丛书

微信小程序开发实战-微课视频版

周文洁　著

清华大学出版社
北　京

内 容 简 介

本书以项目驱动为宗旨，循序渐进、案例丰富，详细介绍了微信小程序的入门基础知识与使用技巧。全书共包含20章，每章均以项目为驱动，将微信小程序的基础知识点分解实现。全书案例由浅入深，从入门篇的创建第一个微信小程序开始，到应用篇的各类API的实现，包括天气查询、口述校史、电子书橱、医疗急救卡、会议邀请函、指南针和手绘时钟等项目；在游戏篇还包含了拼图、推箱子和贪吃蛇游戏；在提高篇引入了小程序云开发的概念，例如带有云数据库的高校新闻网、带有云存储的电子书橱；最后在综合篇给出图片分享社区案例，介绍了第三方Vant Weapp组件库，并结合云开发技术实现了生日管家小程序。

本书包含完整项目案例21个，均在微信Web开发者工具和真机中调试通过，并提供全套项目案例源代码、练习题和视频讲解等电子资源供读者下载。本书适用于微信小程序爱好者、程序设计人员和计算机相关专业学生。

图书在版编目(CIP)数据

微信小程序开发实战：微课视频版/周文洁著. —北京：清华大学出版社，2020
(Web前端开发技术丛书)
ISBN 978-7-302-54163-9

Ⅰ. ①微…　Ⅱ. ①周…　Ⅲ. ①移动终端—应用程序—程序设计　Ⅳ. ①TN929.53

中国版本图书馆CIP数据核字(2019)第248632号

策划编辑：魏江江
责任编辑：王冰飞
封面设计：刘　键
责任校对：李建庄
责任印制：宋　林

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦A座　　**邮　　编**：100084
社 总 机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn，010-83470236
印 装 者：三河市铭诚印务有限公司
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张**：30.5　　**字　　数**：779千字
版　　次：2020年2月第1版　　**印　　次**：2020年2月第1次印刷
印　　数：1～2500
定　　价：89.00元

产品编号：083403-01

前言 FOREWORD

本书以项目驱动为宗旨，循序渐进、案例丰富，详细介绍了微信小程序的入门基础知识与使用技巧。

全书共包含 20 章，可分为以下 6 个部分：

第一部分是入门篇，包括第 1 章和第 2 章。其中第 1 章是开发前的准备，详细讲解如何注册开发者账号和完善信息，以及开发工具的下载与安装；第 2 章是第一个微信小程序，从零开始讲解如何新建项目、真机预览调试等操作，并基于该项目介绍自动生成和手动创建小程序项目的方式。

第二部分是基础篇，包括第 3 章和第 4 章。其中第 3 章是小程序框架，以列表和九宫格两种布局为例，讲解小程序如何使用 flex 布局进行页面规划；第 4 章是小程序组件，以猜数字小游戏为例，介绍表单中文本输入框、按钮等组件的用法。

第三部分是应用篇，包括第 5 章～第 11 章。这 7 个章节分别应用微信小程序中的网络 API、媒体 API、文件 API、数据 API、位置 API、设备 API 以及界面 API，每个 API 均对应一个完整的项目实例，包括天气查询、口述校史、电子书橱、医疗急救卡、会议邀请函、指南针和手绘时钟。

第四部分是游戏篇，包括第 12 章～第 14 章。这 3 个章节基于画布组件和绘图相关 API 分别实现简易版的拼图游戏、推箱子游戏和贪吃蛇游戏。

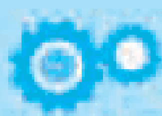
第五部分是提高篇，包括第 15 章～第 18 章。其中第 15 章综合应用之前所学的小程序前端知识开发一款基于模拟数据的高校新闻网小程序；第 16 章和第 17 章对第 15 章的项目进行改造，第 16 章接入自行搭建的服务器后端，形成全栈小程序，第 17 章引入云开发的概念，直接开通云环境即可快速迭代上线；第 18 章对第 7 章的电子书橱项目进行改造，接入云存储功能，可以更方便地下载电子书。

第六部分是综合篇，包括第 19 章和第 20 章。其中第 19 章是对全套云能力的综合应用，实现多用户的图片分享社区；第 20 章结合云能力和第三方组件库 Vant Weapp 快速搭建美观、大方的 UI 界面，实现一款生日管家小程序。

本书有如下几个特点：

(1) 知识全面，循序渐进。

本书首先介绍一些基于小程序框架和组件的基础项目，帮助读者打好基本功；然后正式进入小程序各类应用 API 的相关项目介绍，让读者有针对性地逐步巩固常用小程序 API 的用法；接着介绍 3 款小游戏项目，让读者对未来小游戏的开发学习打下基础。在提高篇补充全栈开发和云开发技术，读者可以根据实际情况自行选择使用第三方服务器或云数据库进行快速开发。最后提供两个综合项目实例，让读者进一步提高对于知识的综合应用能力。



(2) 项目驱动,实用性强。

全书前 11 章将主教材各章节的知识点融入综合项目案例中,帮助读者更好地理解所学知识。第 12 章～第 20 章额外提供了游戏开发、全栈开发、云开发和第三方 UI 组件的应用,具有较强的实用价值,也适合培养读者的动手能力。

(3) 步骤详细,易于理解。

本书思路清晰,知识点循序渐进展开,每章的项目案例均分步骤讲解,读者可以看到从界面设计开始到样式美化以及功能逻辑完成的整个变化过程。读者跟着每章综合案例独立完成开发过程,即可达到小程序前端开发的基本要求。

注: 本书包含 21 个完整项目案例,均在微信 web 开发者工具(目前最新版本为 v1.02.1906141)和真机中调试通过。本书提供 1600 分钟的视频讲解,扫描书中相应章节的二维码可以在线观看学习;本书还提供教学大纲、教学课件、期末试卷、课后拓展作业,扫描封底的课件二维码可以下载。

由于未来微信开发工具软件版本升级和官方文档变更等原因,有可能会导致您在学习时个别功能无法正确显示,如遇此情况请扫描下方二维码查看常见问题汇总文档,我们将会定期更新该文档并告知原因和解决方案。



版本更新



源码+赠送资源

最后感谢清华大学出版社魏江江分社长、王冰飞编辑以及相关工作人员,非常荣幸能有机会与卓越的你们再度合作;感谢家人和朋友给予的关心和大力支持,本书能够完成与你们的鼓励是分不开的;特别感谢刘昕语的支持,让我可以专注于书稿的编写、修订。

愿本书能够对读者学习微信小程序有所帮助,并真诚地欢迎读者批评指正,希望能与读者朋友们共同学习成长,在浩瀚的技术之海不断前行。

作　者

2019 年 10 月

目录 CONTENTS

入 门 篇

基 础 篇

应 用 篇

游 戏 篇

综 合 篇

入门篇

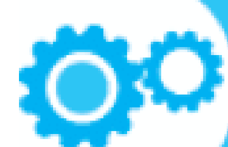
第1章 Chapter 1

开发前的准备

本章内容主要包括开发小程序项目前的两个准备工作，一是注册小程序账号，二是小程序开发工具部署。在注册小程序账号练习中，主要内容包括注册开发者账号、完善小程序信息和管理小程序成员；在小程序开发工具部署练习中，主要内容包括软件的下载与安装、开发者工具的登录和账号切换方法。

本章学习目标

- 掌握开发者账号的注册、信息的完善和成员的管理；
- 掌握开发者工具的下载、安装与登录。



1.1 注册小程序



1.1.1 注册开发者账号

开发者首先需要在微信公众平台上注册一个小程序账号，才能进行后续的代码开发与提交工作。注册步骤如下：

(1) 访问微信公众平台官网首页(mp. weixin. qq. com)，然后单击右上角的“立即注册”按钮进入账号类型选择页面，如图 1-1 所示。



图 1-1　微信公众平台官网首页

（2）在当前页面上选择注册的账号类型为“小程序”，即可进入小程序的正式注册页面，如图 1-2 所示。



图 1-2 账号类型选择页面

（3）小程序的正式注册页面包含 3 个填写步骤，即账号信息、邮箱激活和信息登记。

① 账号信息：在账号信息页面中需要填写邮箱、密码、确认密码、验证码并勾选同意协议条款，如图 1-3 所示。

小程序注册
① 帐号信息 — ② 邮箱激活 — ③ 信息登记
每个邮箱仅能申请一个小程序
已有微信小程序？立即登录
邮箱 testtest@qq.com
作为登录帐号，请填写未被微信公众平台注册，未被微信开放平台注册，未被个人微信号绑定的邮箱
密码 ••••••••••
字母、数字或者英文符号，最短8位，区分大小写
确认密码 ••••••••••
请再次输入密码
验证码 GHTE 换一张
你已阅读并同意《微信公众平台服务协议》及《微信小程序平台服务条款》
注册

图 1-3 小程序账号信息页面

注意该图中所填写的邮箱地址必须符合以下条件：

- 未用于注册过微信公众平台；
- 未用于注册过微信开放平台；

- 未用于绑定过个人微信号的邮箱。

此外，需要注意每个邮箱地址只能申请一个小程序。如果开发者当前暂时没有符合条件的邮箱，建议先去申请一个新的邮箱再来继续小程序的账号注册。

全部填写完成并勾选同意协议条款后，单击最下方的“注册”按钮提交账号信息。

② 邮箱激活：在提交注册后会看到邮箱激活提醒，此时页面效果如图 1-4 所示。



图 1-4　邮箱激活提醒

登录对应的注册邮箱查看激活邮件，如图 1-5 所示。



图 1-5　小程序激活邮件

单击邮件正文中的链接地址会跳转到微信平台页面完成账号激活。

③ 信息登记：邮箱账号激活完成后就进入了信息登记页面，如图 1-6 所示。

其中注册国家/地区保持默认内容“中国大陆”，然后根据实际情况进行主体类型的选择。目前小程序允许注册的主体类型共有 5 种，包括个人、企业、政府、媒体以及其他组织，详情见表 1-1。

小程序注册

① 帐号信息 —— ② 邮箱激活 —— ③ 信息登记

用户信息登记

微信公众平台致力于打造真实、合法、有效的互联网平台。为了更好的保障你和广大微信用户的合法权益，请你认真填写以下登记信息。
为表述方便，本服务中，"用户"也称为"开发者"或"你"。

用户信息登记审核通过后：
1. 你可以依法享有本微信公众帐号所产生的权利和收益；
2. 你将对本微信公众帐号的所有行为承担全部责任；
3. 你的注册信息将在法律允许的范围内向微信用户展示；
4. 人民法院、检察院、公安机关等有权机关可向腾讯依法调取你的注册信息等。

请确认你的微信公众帐号主体类型属于政府、媒体、企业、其他组织、个人，并请按照对应的类别进行信息登记。
点击查看微信公众平台信息登记指引。

注册国家/地区 中国大陆

主体类型 如何选择主体类型？

个人 企业 政府 媒体 其他组织

个人类型包括：由自然人注册和运营的公众帐号。
帐号能力：个人类型暂不支持微信认证、微信支付及高级接口能力。

图 1-6 小程序信息登记页面

表 1-1 小程序账号主体类型介绍

账号主体类型	解　　释
个人	必须是 18 岁以上的微信实名用户，并且具有国内身份信息
企业	企业、分支机构、个体工商户或企业相关品牌
政府	国内各级/各类政府机构、事业单位、具有行政职能的社会组织等，目前主要覆盖公安机构、党团机构、司法机构、交通机构、旅游机构、工商税务机构、市政机构等
媒体	报纸、杂志、电视、电台、通讯社、其他等
其他组织	不属于政府、媒体、企业或个人的其他类型

由于本书为个人开发者小程序入门，所以选择“个人”类型。企业类型账号注册需要企业缴费认证，而政府、媒体或其他组织账号注册需要通过微信验证主体单位的身份，对于这几种类型暂不介绍，后续可以由开发者自行申请这些主体类型。

选择“个人”类型之后，在页面下方会自动出现主体信息登记表单，如图 1-7 所示。

开发者需要如实填写身份证姓名、身份证号码、管理员手机号码(一个手机号码只能注册 5 个小程序)，然后单击“获取验证码”按钮等待手机短信。在收到的短信中会提供一个 6 位验证码，如图 1-8 所示。

注意验证码必须在 10 分钟之内填写，否则会失效，需要重新获取。填写完成后在下方的管理员身份验证栏中会自动出现一个二维码，如图 1-9 所示。

此时需要管理员用本人微信扫描页面上提供的二维码来进行身份确认。这种验证方式是免费的，不会扣取任何费用。扫码后手机微信会自动跳转到微信验证页面，如图 1-10 所示。

检查微信验证页面上所显示的姓名和身份证号码，确认无误后点击“确定”按钮会提示身份验证成功，如图 1-11 所示。

此时该微信号就会被登记为管理员微信号，并且计算机端的网页画面也将同步提示“身份验证成功”，如图 1-12 所示。

主体类型　如何选择主体类型？

个人　企业　政府　媒体　其他组织

个人类型包括：由自然人注册和运营的公众帐号。
帐号能力：个人类型暂不支持微信认证、微信支付及高级接口能力。

主体信息登记

身份证姓名
信息审核成功后身份证姓名不可修改；如果名字包含分隔号"·"，请勿省略。

身份证号码
请输入您的身份证号码。一个身份证号码只能注册5个小程序。

管理员手机号码　获取验证码
请输入您的手机号码，一个手机号码只能注册5个小程序。

短信验证码　无法接收验证码？
请输入手机短信收到的6位验证码

管理员身份验证　请先填写管理员身份信息

继续

图 1-7　小程序信息登记页面

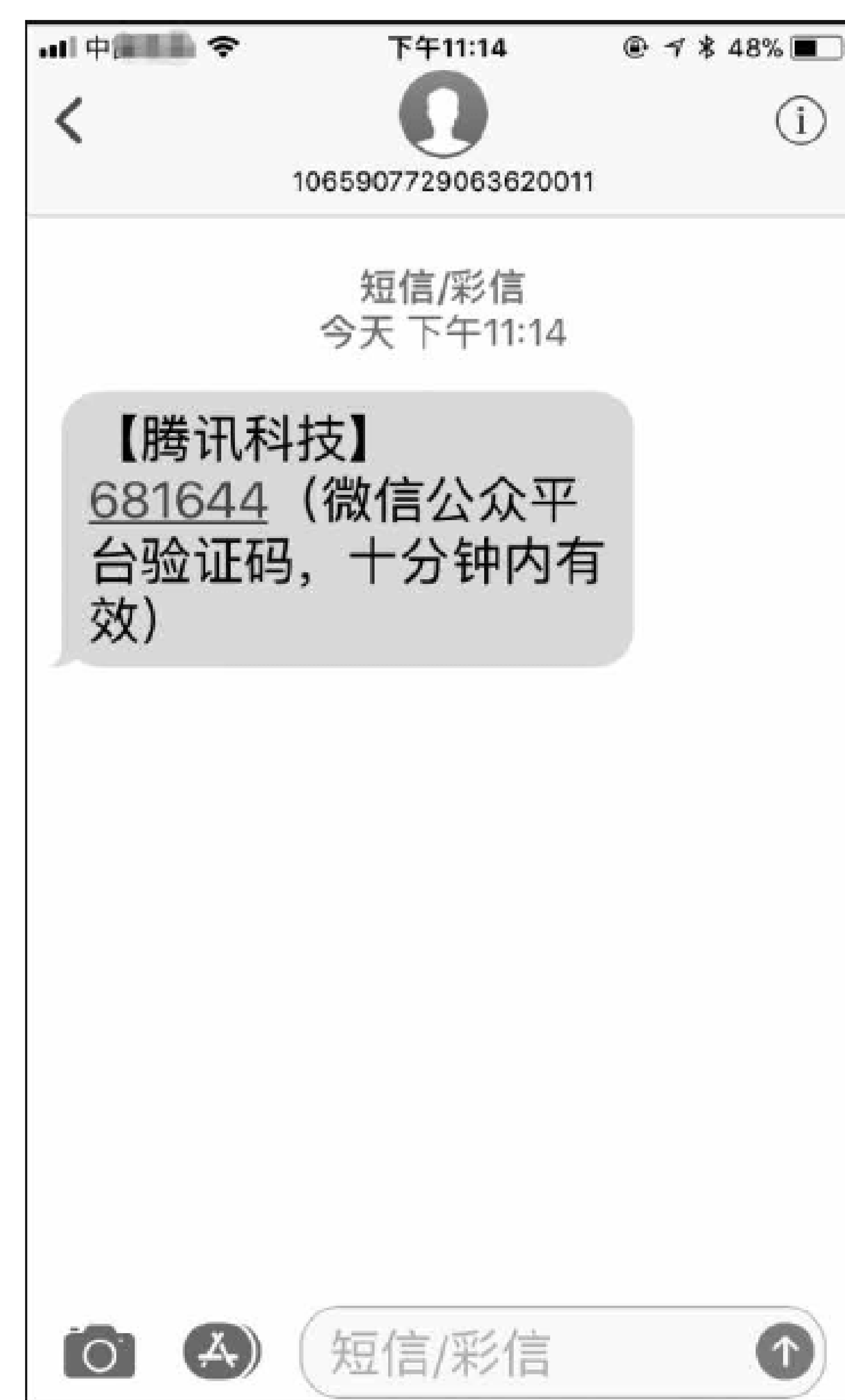


图 1-8　小程序验证码短信

主体类型　如何选择主体类型？

个人　企业　政府　媒体　其他组织

个人类型包括：由自然人注册和运营的公众帐号。
帐号能力：个人类型暂不支持微信认证、微信支付及高级接口能力。

主体信息登记

身份证姓名　周
信息审核成功后身份证姓名不可修改；如果名字包含分隔号"·"，请勿省略。

身份证号码　3402198 11082
请输入您的身份证号码。一个身份证号码只能注册5个小程序。

管理员手机号码　153 021　17秒后可重发
请输入您的手机号码，一个手机号码只能注册5个小程序。

短信验证码　681644　无法接收验证码？
请输入手机短信收到的6位验证码

管理员身份验证
请用管理员本人微信扫描二维码。本验证方式不扣除任何费用。注册后，扫码的微信号将成为该帐号的管理员微信号。

继续

图 1-9　管理员身份验证栏中出现二维码

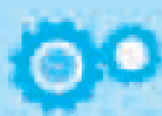




图 1-10 手机微信验证身份确认页面



图 1-11 手机微信验证成功页面



图 1-12 管理员身份验证成功

然后单击下方的“继续”按钮进行下一步，系统会弹出一个提示框让开发者做最后的确认，如图 1-13 所示。



图 1-13 主体信息确认提示框

单击“确定”按钮完成主体信息确认，会出现如图 1-14 所示的内容。



图 1-14 信息提交成功提示框

当前可以直接单击“前往小程序”按钮进入小程序管理页面，此时账号是默认登录后的状态，可以直接进行小程序的后续管理工作，如图 1-15 所示。



图 1-15 小程序管理页面

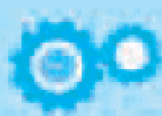
现在小程序的账号注册就全部完成了，之后可以访问微信公众平台(mp. weixin. qq. com)手动输入账号和密码登录进入小程序管理页面。

1.1.2 完善小程序信息

账号注册完成后还需要完善小程序的基本信息，如表 1-2 所示。

表 1-2 小程序基本信息内容介绍

填写内容	填写要求	修改次数
小程序名称	小程序名称需要控制在 4～30 个字符，并且不得与平台内已经存在的其他账号名称重复	发布前有两次改名机会； 两次改名机会用完后，必须先发布再通过微信认证改名
小程序头像	图片格式只能是 PNG、BMP、JPEG、JPG、GIF 中的一种，并且文件不得大于 2MB。注意，头像图片不允许涉及政治敏感与色情内容。图片最后会被切割为圆形效果	每个月可修改 5 次



续表

填写内容	填写要求	修改次数
小程序介绍	字数必须控制在4～120个字符，介绍内容不得含有国家相关法律、法规禁止的内容	每个月可以申请修改5次
小程序服务类目	服务类目分为两级，每一级都必须填写，不可以为空。服务类目不得少于1个，不得多于5个。特殊行业需要额外提供资质证明	每个月可以修改1次

注意：一个中文字占两个字符。

1 小程序名称

小程序名称的长度需要控制在4～30个字符，其中一个中文字占两个字符。在小程序发布前有两次改名机会。两次改名机会用完后，必须先发布再通过微信认证改名。

由于小程序名称不允许与平台内已经存在的其他账号名称重复，所以在填写好之后可以先自测一下是否符合要求，单击右侧的"检测"按钮即可进行验证。

如果该名称已经与其他公众号名称重复，会出现失败提示，如图1-16所示。

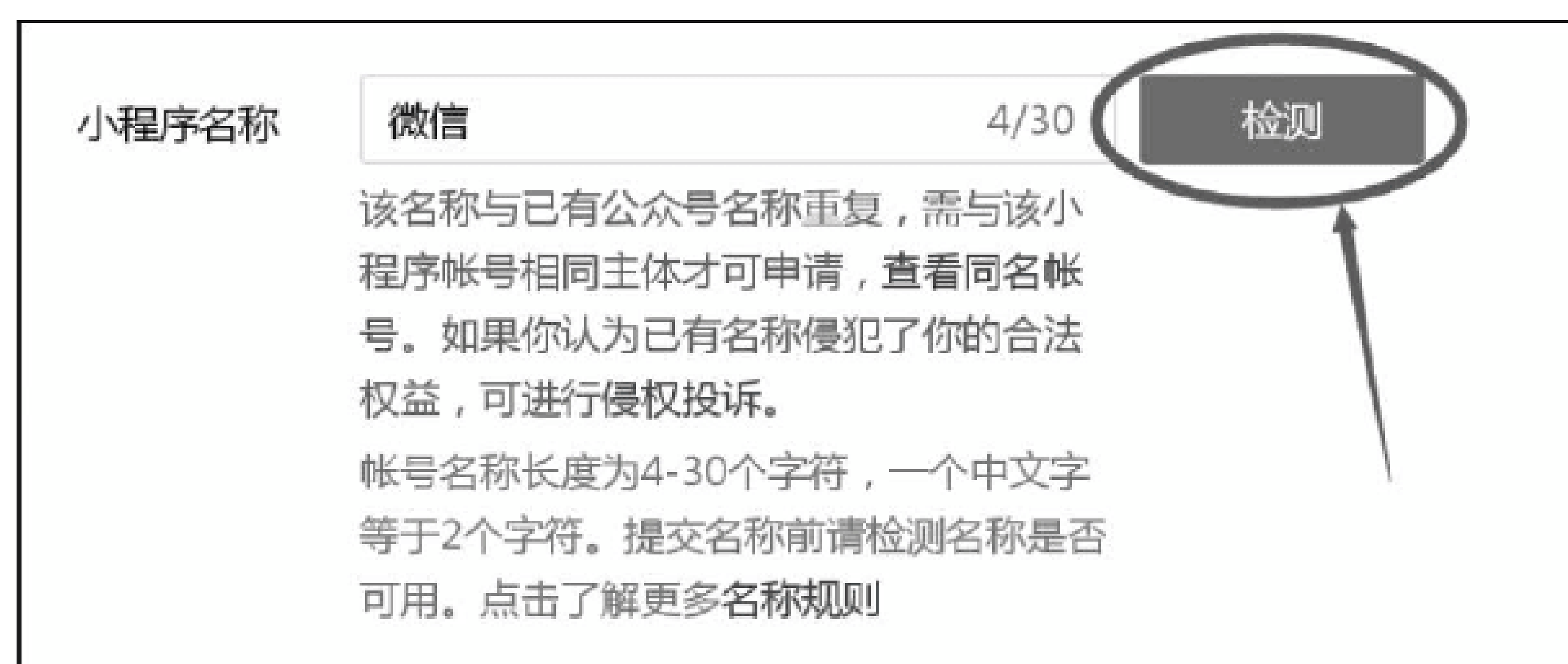


图1-16 小程序名称检测失败提示

如果该名称没有被占用，检测后会显示"你的名字可以使用"字样，如图1-17所示。

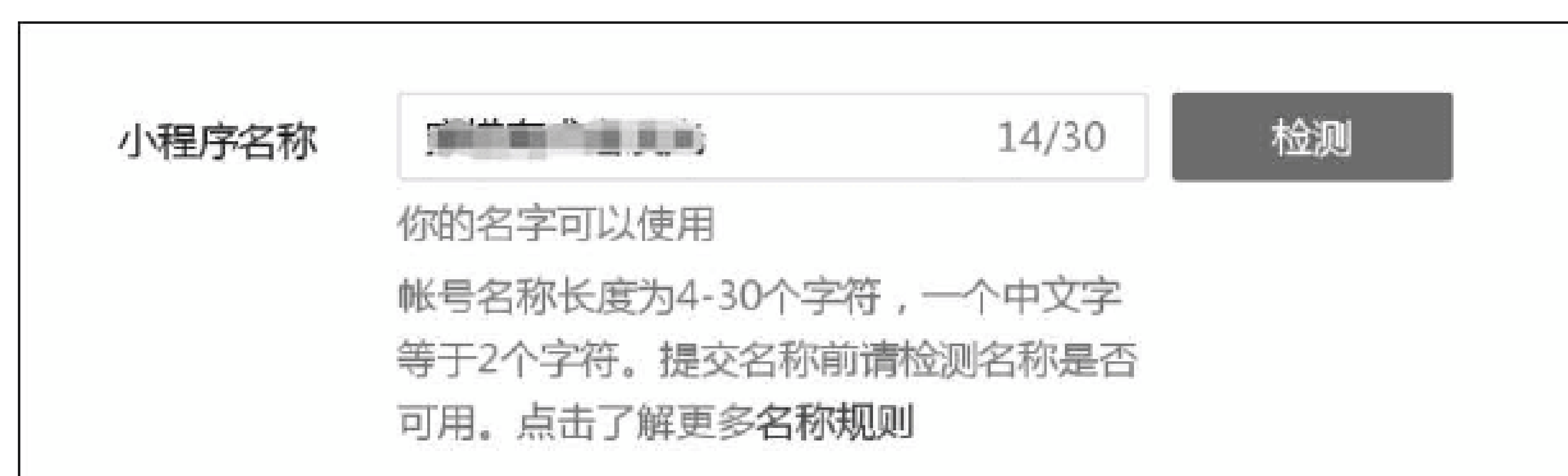


图1-17 小程序名称检测成功提示

该图就表示该名称允许使用，接下来就可以上传图片了。

2 小程序头像

小程序头像也就是小程序最终显示的logo图标，图片最后会被切割为圆形效果。头像图片的格式只能是PNG、BMP、JPEG、JPG、GIF中的一种，并且文件大小不得大于2MB。注意，头像图片不允许涉及政治敏感与色情内容。头像图片每个月可修改5次。

单击"选择图片"按钮即可选择图片进行上传，如图1-18所示。

根据官方提示，建议上传PNG格式的图片并且图片尺寸为144像素×144像素，以保证最佳效果。

3 小程序介绍

小程序介绍可以由开发者自由填写关于小程序功能的描述，注意介绍内容不得含有国家



图 1-18　小程序头像上传

相关法律、法规禁止的内容。小程序介绍的内容每个月可以申请修改 5 次。

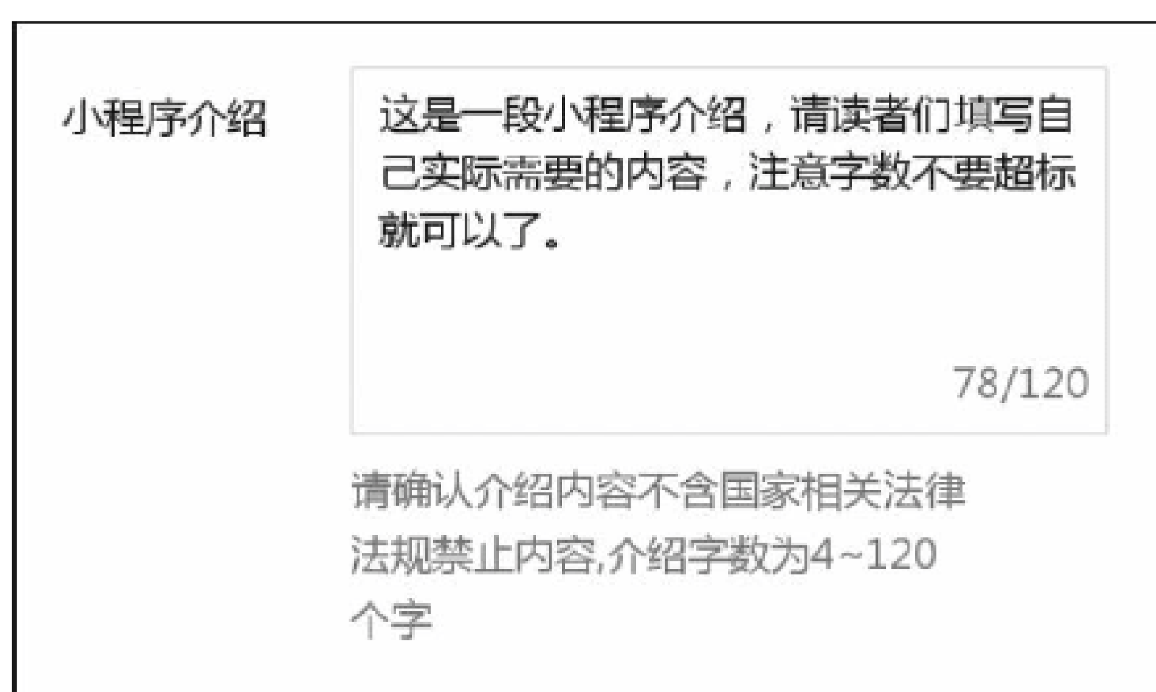


图 1-19　小程序介绍

小程序介绍对应的字数必须控制在 4～120 个字符，文本框带有自动检测字数的功能，如图 1-19 所示。

4 小程序服务类目

小程序服务类目指的是小程序主要内容所属的服务范围。特殊行业需要额外提供资质证明。服务类目每个月只可以修改 1 次。

服务类目的下拉表单分为两级，每一级都必须填写，不可以为空，如图 1-20 所示。



图 1-20　小程序服务类目的二级选项

如果有多个服务范围需要追加，可以单击右侧的＋号进行添加，如图 1-21 所示。

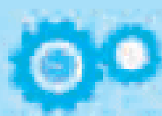


图 1-21　小程序服务类目的追加

如果需要去掉多余的服务范围，将鼠标移动到需要删除的服务类目上，然后单击右侧出现的一号进行删除，如图 1-22 所示。



图 1-22　小程序服务类目的删除



注意：小程序的服务类目最少必须有 1 个，最多只能有 5 个。

全部填写完毕后，就可以单击最下方的“提交”按钮提交小程序的基本信息，完成后可以看到如图 1-23 所示的界面。



图 1-23　小程序信息填写完成

此时就可以单击“添加开发者”按钮进行小程序成员的管理了。

1.1.3　管理小程序成员

除了管理员以外，还可以为小程序追加其他项目成员。具有管理员身份的开发者登录后可以在小程序管理后台统一管理项目成员，并为他们分别设置对应的权限。

1 成员类型说明

管理员可以为小程序添加开发者、体验者以及其他权限的项目成员。

项目成员可以被分配的不同权限如下。

- 开发者：可以使用微信 web 开发者工具进行小程序开发，也可以预览开发版小程序在手机端的效果；
- 体验者：可以在手机端使用体验版小程序；
- 登录：无须管理员确认即可登录小程序管理后台；
- 数据分析：可以使用小程序数据分析功能查看小程序数据；
- 开发管理：拥有小程序提交审核、发布和回退权限；
- 开发设置：拥有设置小程序服务器域名、消息推送以及扫描普通链接二维码打开小程序的权限；
- 暂停服务设置：拥有暂停小程序线上服务的权限。

2 成员人数限制

个人类型的小程序允许管理员添加 15 个开发者，其中 5 个开发者、10 个体验者。其他类型的小程序开发者的数量限制如下。

- 未认证未发布组织类型：30 人；
- 已认证未发布/未认证已发布组织类型：60 人；

- 已认证已发布组织类型：90 人。

3 成员变更说明

每个小程序的管理员与项目成员都是允许变更的。需要注意的是，每个微信号作为项目成员最多可以参与到 50 个小程序中。



1.2 小程序开发工具



在完成了准备工作之后就可以进行小程序开发了。小程序具有官方提供的专属开发工具——微信 web 开发者工具。

1.2.1 软件的下载与安装

开发者登录小程序管理页面后台，然后单击右上角菜单栏中的“开发”选项即可切换到小程序开发工具的下载页面，也可以直接通过 URL 地址访问下载页面，例如：

https://mp.weixin.qq.com/debug/wxadoc/dev/devtools/download.html

在该页面中需要根据自己计算机的操作系统的类型选择对应的下载地址。目前提供的 3 种下载地址与计算机操作系统对应的关系如下。

- Windows 64：Windows 64 位操作系统；
- Windows 32：Windows 32 位操作系统；
- Mac：Mac 操作系统。

注意： 本书下载的是 Windows 64 版本，读者请根据实际情况选择对应的软件进行下载。

以 Windows 64 版本为例，下载完成后会获得一个 EXE 应用程序文件，如图 1-24 所示。

名称	修改日期	类型	大小
wechat_devtools_1.02.1801081_x64.exe	2018/1/11 20:20	应用程序	75,765 KB

图 1-24 微信 web 开发者工具的安装文件

该图中的“1.02.1801081”为软件版本号，“_x64”表示为 Windows 64 位版本软件。读者可以根据文件名再次确认是否下载了正确的版本。

确认无误后可以双击该文件进行开发者工具的安装，安装过程如图 1-25 所示。

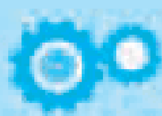
安装完成后的页面如图 1-26 所示。

1.2.2 开发者工具的登录

微信 web 开发者工具要使用开发者微信账号登录后才可以进行小程序开发。

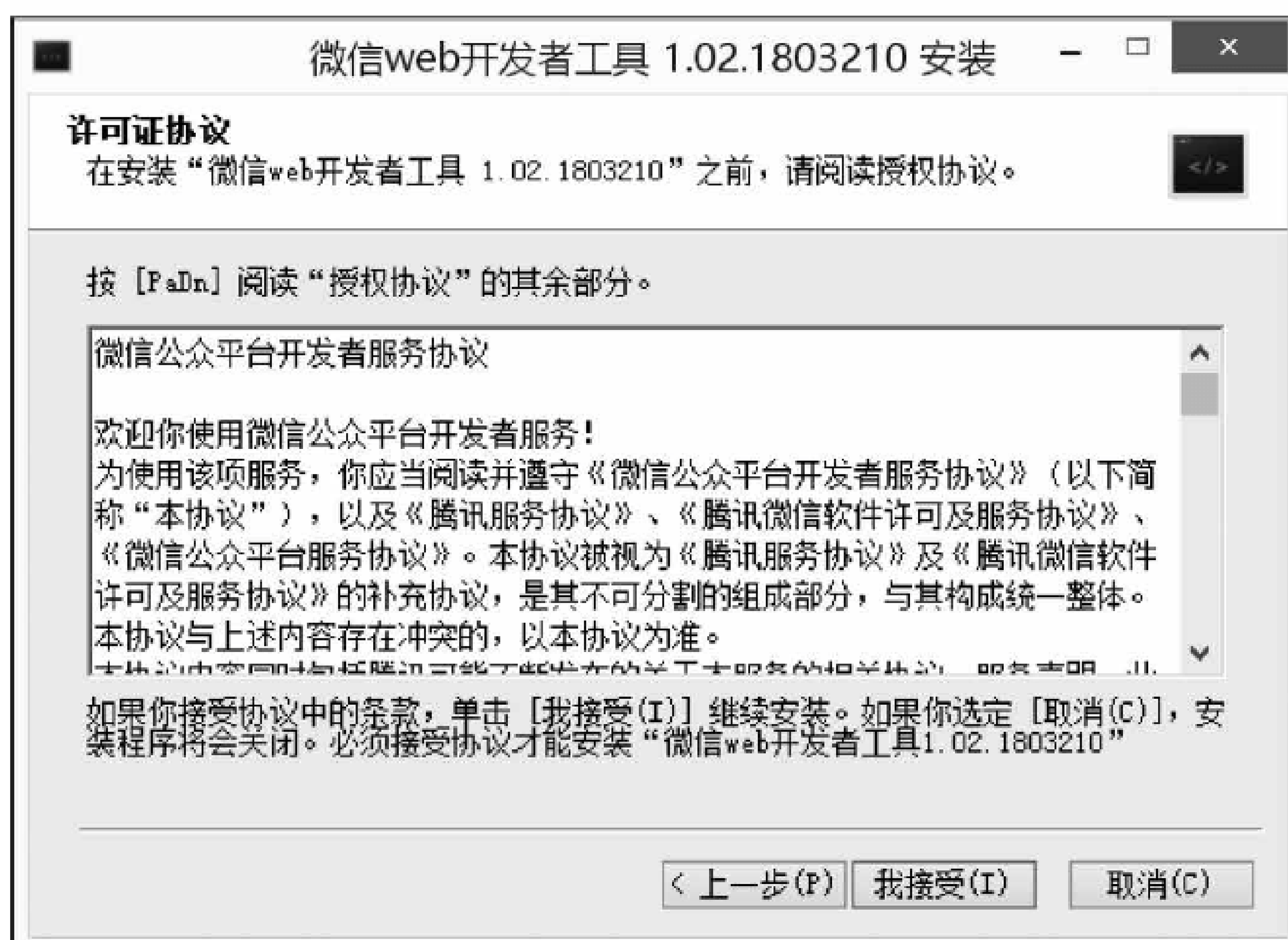
1 开发者身份验证

与一般手动输入账号和密码的流程不同，微信 web 开发者工具是使用微信扫描二维码的方式验证开发者身份。在计算机端双击微信开发者工具图标，会弹出微信开发者工具登录页面，如图 1-27 所示。

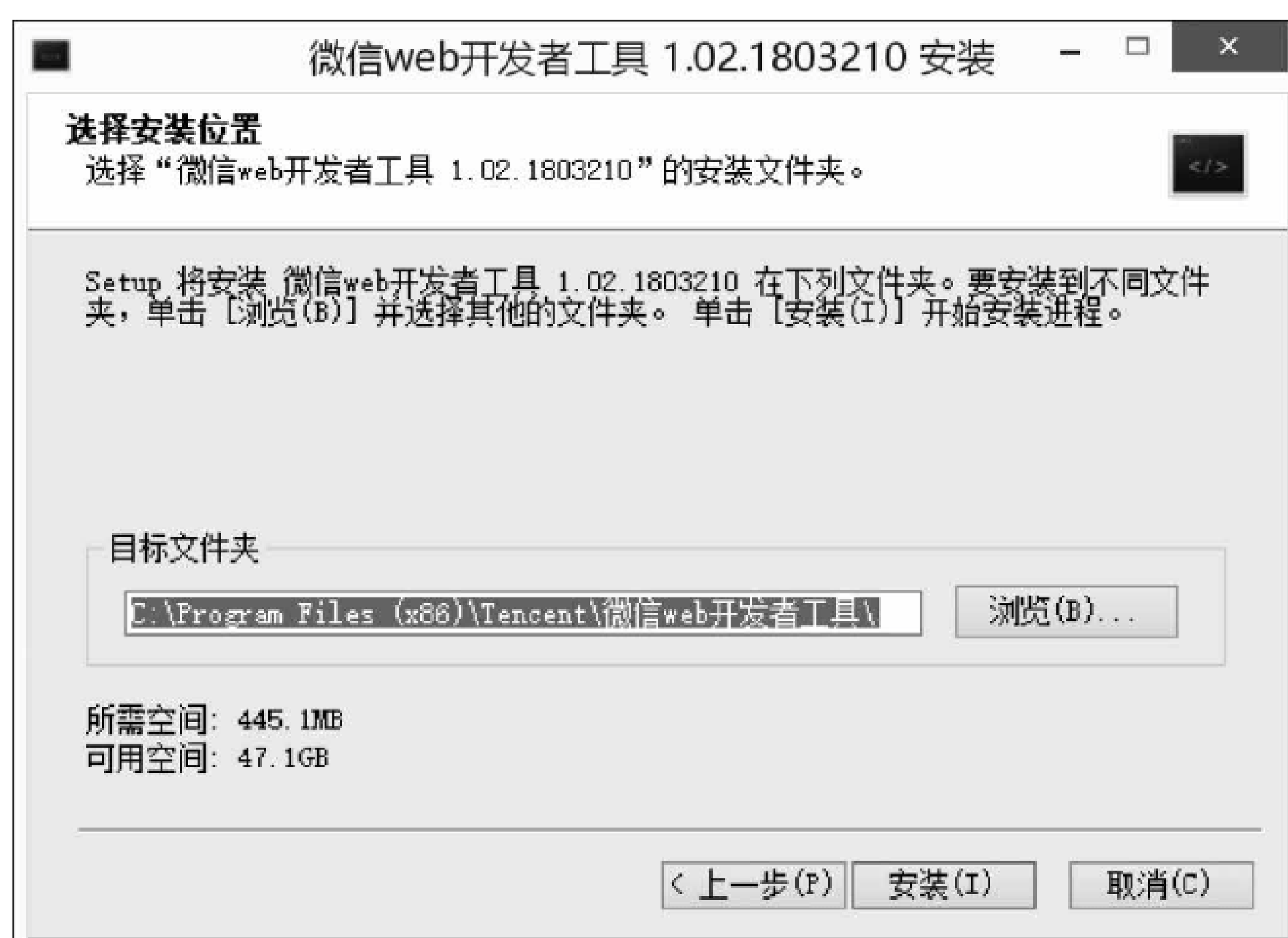
 



(a) 进入安装向导



(b) 授权许可证协议



(c) 选择安装位置

图 1-25　微信 web 开发者工具的安装过程



(d) 正在安装

图 1-25　（续）



图 1-26　微信 web 开发者工具的安装完成界面



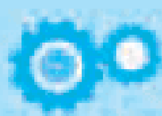
图 1-27　微信开发者工具登录页面

开发者通过用手机微信扫描计算机端的二维码来确认身份，手机端的效果如图 1-28 所示。

其中图 1-28(a)为点击手机微信右上角的加号出现的下拉菜单，点击其中的“扫一扫”选项进行二维码扫描；图 1-28(b)为扫码成功后跳转的提示页面，用户点击“确认登录”按钮即可登录并使用微信开发者工具。

在这个过程中，计算机端的页面变化如图 1-29 所示。

其中图 1-29(a)为手机微信扫码成功后出现的提示页面，注意该二维码是动态变化的，并且长时间不扫描会超时过期；图 1-29(b)显示的菜单页面是当开发者在手机微信上点击“确认登录”按钮后才会出现的，此时就可以正式进行小程序开发了。





(a) 手机微信“扫一扫”选项



(b) 扫码后手机微信出现确认提示

图 1-28 手机微信扫码过程



(a) 扫码成功的提示页面

(b) 确认登录后的菜单页面

图 1-29 扫码确认过程中计算机端的页面变化

2 开发者账号切换

微信开发者工具允许在同一台计算机上切换不同的开发者。如果用户登录后发现需要更换账号，可以选择菜单页面左下角的“注销”选项回到二维码扫描页面，使用其他开发者微信账号重新扫码登录，如图 1-30 所示。

第 2 章将正式进行小程序项目的创建和开发。



(a) 选择“注销”选项　(b) 重新回到二维码扫描页面

图 1-30　开发者账号切换

第2章 Chapter 2

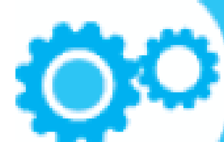


第一个微信小程序

本章内容主要包含两个创建小程序的项目实例，一是使用快速启动模板创建小程序，二是手动从零开始创建小程序。这两个项目的主要功能都是点击按钮获取当前微信用户的头像和昵称。

本章学习目标

- 学习使用快速启动模板创建小程序的方法；
- 学习不使用模板手动创建小程序的方法。



2.1 自动生成小程序



为方便初学者入门，微信开发者工具提供了快速启动模板帮助开发者自动生成代码，从而创建完成第一个示例小程序。下面使用开发者工具正式创建第一个微信小程序。

2.1.1 项目创建



视频讲解

双击微信开发者工具图标，管理员或开发者使用微信扫描二维码后进入菜单画面。选择菜单中的“小程序”选项进入小程序项目管理页面，如图 2-1 所示。



图 2-1 新建小程序项目

此时开发者依次填写项目名称、目录和 AppID 就可以新建一个小程序项目。填写时的注意事项如下。

- 项目名称：由开发者自定义一个项目名称，该名称仅供开发者工具管理使用，不是小程序被用户看到的名字。
- 目录：项目文件存放的路径地址，可以单击输入框右侧的箭头按钮在计算机盘符中选择指定的目录地址。该地址可以由开发者自定义。
- AppID：管理员在微信公众平台上注册的小程序 ID。

其中小程序的 AppID 可以登录微信公众平台(https://mp.weixin.qq.com)查看，具体查看步骤是单击左侧菜单中的“开发”选项，切换至“开发设置”面板，查看“开发者 ID”下方的 AppID(小程序 ID)，如图 2-2 所示。



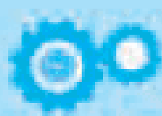
图 2-2 查看小程序 ID

将该小程序 ID 复制、粘贴到图 2-1 所示的 AppID 输入框中。

填完以后的效果如图 2-3 所示。



图 2-3 小程序项目填写效果示意图



其中 AppID 必须填实际的小程序 ID,否则部分功能将无法使用。如果开发者暂时条件受限无法注册申请小程序 ID,可以使用 AppID 输入框下方的“测试号”来体验小程序。

项目目录选择空白文件夹 tmplDemo,则开发者工具会默认自动生成代码形成一个简单的小程序效果以供初学者入门学习。“后端服务”选择“不使用云服务”,其他选项保持默认不变。

填写完毕后单击“新建”按钮完成操作,此时会跳转到开发页面,如图 2-4 所示。

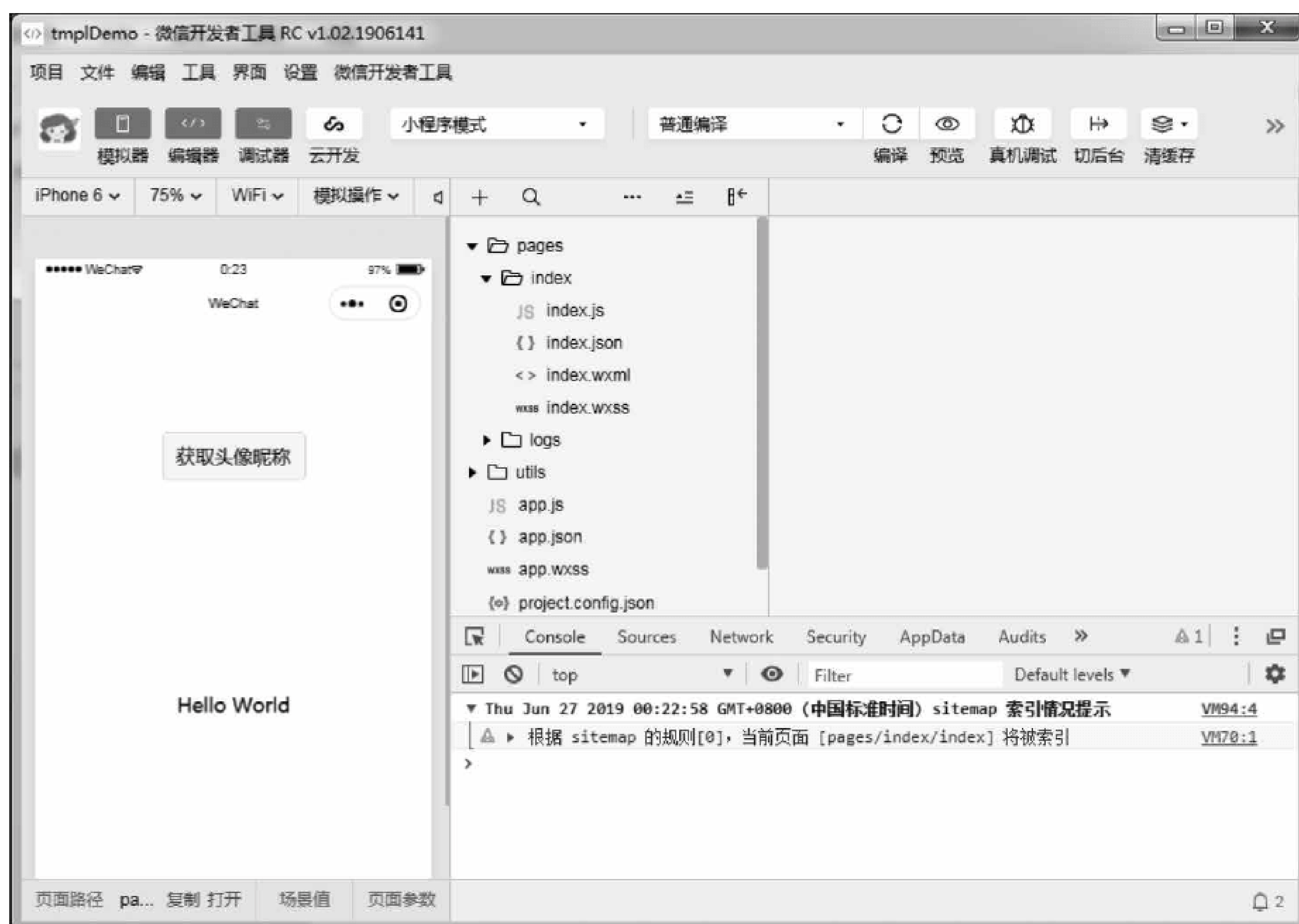


图 2-4 小程序项目开发页面

图中左边就是手机预览效果图,由图可见目前能显示出来一个“获取头像昵称”按钮和一个“Hello World”文本,这与手机运行的效果完全相同。

用户可以直接在上面用鼠标左键单击来模拟手指在手机屏幕上的触摸效果,如图 2-5 所示。



(a) 用鼠标左键单击按钮　(b) 单击“允许”按钮　(c) 最终显示效果

图 2-5 小程序项目单击效果

其中图 2-5(a)显示的是用鼠标左键单击按钮来模拟手机上的触摸点击效果；图 2-5(b)为单击之后弹出的微信授权信息，只有单击“允许”按钮才可以获得数据；图 2-5(c)为最终显示效果，由该图可见小程序项目已经成功地获取了开发者头像和昵称信息。

2.1.2　真机预览



视频讲解

除了可以直接在计算机端使用鼠标模拟手机触屏的点击效果外，还可以直接在真机上进行程序预览。用鼠标左键单击“预览”按钮，即可自动生成一个预览二维码，如图 2-6 所示。



图 2-6　小程序项目生成预览二维码

此时用手机微信扫描图 2-6 中的二维码即可进行真机预览，如图 2-7 所示。

由图可见，画面效果基本上和计算机端的预览图一致。另外，预览所用的二维码不是永久有效，用户要注意它的过期时间，一旦过期需要重新单击“预览”按钮生成新的预览二维码。



图 2-7　小程序项目的真机预览效果

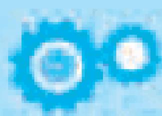
2.1.3　完整代码展示

说明：本项目的全部代码都是由开发者工具自动生成的，读者暂时不需要完全理解，待未来深入学习后再回头来看。

1 配置文件展示

完整的 app.json 代码如下：

```
1.  {
2.    "pages": [
3.      "pages/index/index",
```



```
4.     "pages/logs/logs"
5.   ],
6.   "window": {
7.     "backgroundTextStyle": "light",
8.     "navigationBarBackgroundColor": "#FFF",
9.     "navigationBarTitleText": "WeChat",
10.    "navigationBarTextStyle": "black"
11.  },
12.  "sitemapLocation": "sitemap.json"
13. }
```

完整的 app.wxss 代码如下：

```
1. /** app.wxss **/
2. .container {
3.   height: 100%;
4.   display: flex;
5.   flex-direction: column;
6.   align-items: center;
7.   justify-content: space-between;
8.   padding: 200rpx 0;
9.   box-sizing: border-box;
10. }
```

完整的 app.js 代码如下：

```
1. //app.js
2. App({
3.   onLaunch: function() {
4.     //展示本地存储能力
5.     var logs = wx.getStorageSync('logs') || []
6.     logs.unshift(Date.now())
7.     wx.setStorageSync('logs', logs)
8.
9.     //登录
10.    wx.login({
11.      success: res => {
12.        //发送 res.code 到后台换取 openId、sessionKey、unionId
13.      }
14.    })
15.    //获取用户信息
16.    wx.getSetting({
17.      success: res => {
18.        if (res.authSetting['scope.userInfo']) {
19.          //已经授权,可以直接调用 getUserInfo 获取头像昵称,不会弹框
20.          wx.getUserInfo({
21.            success: res => {
22.              //可以将 res 发送给后台解码出 unionId
23.              this.globalData.userInfo = res.userInfo
24.
25.              //由于 getUserInfo 是网络请求,可能会在 Page.onLoad 之后才返回
26.              //所以此处加入 callback 来防止这种情况
27.              if (this.userInfoReadyCallback) {
28.                this.userInfoReadyCallback(res)
29.              }
30.            }
31.          })
32.        }
```

```
33.         }
34.       })
35.     },
36.     globalData: {
37.       userInfo: null
38.     }
39.  })
```

2 公共文件展示

完整的公共JS(utils/util.js)代码如下:

```
1.  const formatTime = date => {
2.    const year = date.getFullYear()
3.    const month = date.getMonth() + 1
4.    const day = date.getDate()
5.    const hour = date.getHours()
6.    const minute = date.getMinutes()
7.    const second = date.getSeconds()
8.
9.    return [year, month, day].map(formatNumber).join('/') + ' ' + [hour, minute,
second].map(formatNumber).join(':')
10. }
11.
12. const formatNumber = n => {
13.   n = n.toString()
14.   return n[1] ? n : '0' + n
15. }
16.
17. module.exports = {
18.   formatTime: formatTime
19. }
```

3 页面文件展示

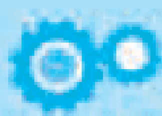
1) index 页面

完整的 WXML(pages/index/index.wxml)代码如下:

```
1.  <!-- index.wxml -->
2.  <view class="container">
3.    <view class="userinfo">
4.      <button wx:if="{{!hasUserInfo && canIUse}}" open-type="getUserInfo"
bindgetuserinfo="getUserInfo"> 获取头像昵称 </button>
5.      <block wx:else>
6.        <image bindtap="bindViewTap" class="userinfo-avatar" src="{{userInfo.
avatarUrl}}" mode="cover"></image>
7.        <text class="userinfo-nickname">{{userInfo.nickName}}</text>
8.      </block>
9.    </view>
10.   <view class="usermotto">
11.     <text class="user-motto">{{motto}}</text>
12.   </view>
13. </view>
```

完整的 WXSS(pages/index/index.wxss)代码如下:

```
1.  /** index.wxss **/
2.  .userinfo {
3.    display: flex;
```

```
4.     flex-direction: column;
5.     align-items: center;
6.   }
7.
8.   .userinfo-avatar {
9.     width: 128rpx;
10.    height: 128rpx;
11.    margin: 20rpx;
12.    border-radius: 50%;
13.  }
14.
15.  .userinfo-nickname {
16.    color: #AAA;
17.  }
18.
19.  .usermotto {
20.    margin-top: 200px;
21.  }
```

完整的 JS(pages/index/index.js)代码如下：

```
1.   //index.js
2.   //获取应用实例
3.   const app = getApp()
4.
5.   Page({
6.     data: {
7.       motto: 'Hello World',
8.       userInfo: {},
9.       hasUserInfo: false,
10.      canIUse: wx.canIUse('button.open-type.getUserInfo')
11.    },
12.    //事件处理函数
13.    bindViewTap: function() {
14.      wx.navigateTo({
15.        url: '../logs/logs'
16.      })
17.    },
18.    onLoad: function() {
19.      if (app.globalData.userInfo) {
20.        this.setData({
21.          userInfo: app.globalData.userInfo,
22.          hasUserInfo: true
23.        })
24.      } else if (this.data.canIUse){
25.        //由于 getUserInfo 是网络请求，可能会在 Page.onLoad 之后才返回
26.        //所以此处加入 callback 来防止这种情况
27.        app.userInfoReadyCallback = res => {
28.          this.setData({
29.            userInfo: res.userInfo,
30.            hasUserInfo: true
31.          })
32.        }
33.      } else {
34.        //在没有 open-type=getUserInfo 版本时的兼容处理
35.        wx.getUserInfo({
36.          success: res => {
37.            app.globalData.userInfo = res.userInfo
```

```
38.             this.setData({
39.               userInfo: res.userInfo,
40.               hasUserInfo: true
41.             })
42.           }
43.         })
44.       }
45.     },
46.     getUserInfo: function(e) {
47.       console.log(e)
48.       app.globalData.userInfo = e.detail.userInfo
49.       this.setData({
50.         userInfo: e.detail.userInfo,
51.         hasUserInfo: true
52.       })
53.     }
54.   })
```

完整的 JSON(pages/index/index.json)代码如下：

```
1.  {
2.    "usingComponents": {}
3.  }
```

2）logs 页面

完整的 WXML(pages/logs/logs.wxml)代码如下：

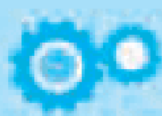
```
1.  <!-- logs.wxml -->
2.  <view class="container log-list">
3.    <block wx:for="{{logs}}" wx:for-item="log">
4.      <text class="log-item">{{index + 1}}. {{log}}</text>
5.    </block>
6.  </view>
```

完整的 WXSS(pages/logs/logs.wxss)代码如下：

```
1.  .log-list {
2.    display: flex;
3.    flex-direction: column;
4.    padding: 40rpx;
5.  }
6.  .log-item {
7.    margin: 10rpx;
8.  }
```

完整的 JS(pages/logs/logs.js)代码如下：

```
1.  //logs.js
2.  const util = require('../../utils/util.js')
3.
4.  Page({
5.    data: {
6.      logs: []
7.    },
8.    onLoad: function() {
9.      this.setData({
10.       logs: (wx.getStorageSync('logs') || []).map(log => {
11.         return util.formatTime(new Date(log))
```

```
12.         })
13.       })
14.     }
15.  })
```

完整的 JSON(pages/logs/logs.json)代码如下：

```
1.  {
2.    "navigationBarTitleText": "查看启动日志",
3.    "usingComponents": {}
4.  }
```



2.2 手动创建小程序



在熟悉了小程序的目录结构和微信开发者工具的基本使用方法后，读者不妨尝试自己动手创建第一个小程序。

2.2.1 项目创建



视频讲解

本项目创建选择空白文件夹 myDemo，效果如图 2-8 所示。

此时仍然会自动生成模板代码，2.2.2 节将介绍如何手动修改页面配置文件。



图 2-8 小程序项目填写效果示意图

2.2.2 页面配置



视频讲解

1 创建页面文件

项目创建完毕后，在根目录中会生成文件夹 pages 用于存放页面文件。

一般来说首页默认命名为index,表示小程序运行的第一个页面;其他页面名称可以自定义。本项目只需要保留首页(index)即可。

具体操作如下:

(1) 将app.json文件内pages属性中的"pages/logs/logs"删除,并删除上一行末尾的逗号。

(2) 按快捷键Ctrl+S保存当前修改。

2 删除和修改文件

具体操作如下:

(1) 删除utils文件夹及其内部所有内容。

(2) 删除pages文件夹下的logs目录及其内部所有内容。

(3) 删除index.wxml和index.wxss中的全部代码。

(4) 删除index.js中的全部代码,并且输入关键词"page"找到第二个选项按回车键让其自动补全函数(如图2-9所示)。

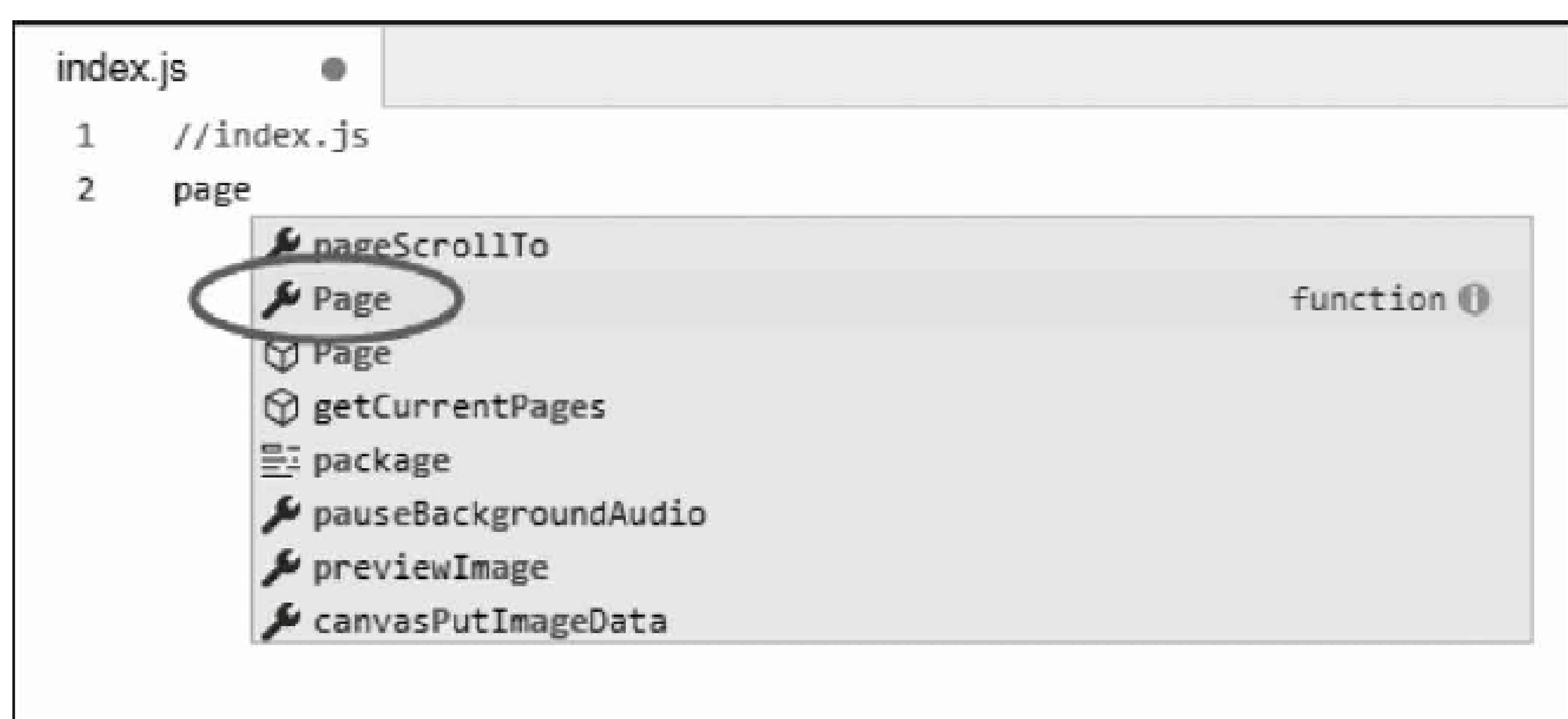


图2-9 输入关键词创建Page函数

(5) 删除app.wxss中的全部代码。

(6) 删除app.js中的全部代码,并且输入关键词"app"找到第一个选项按回车键让其自动补全函数(如图2-10所示)。

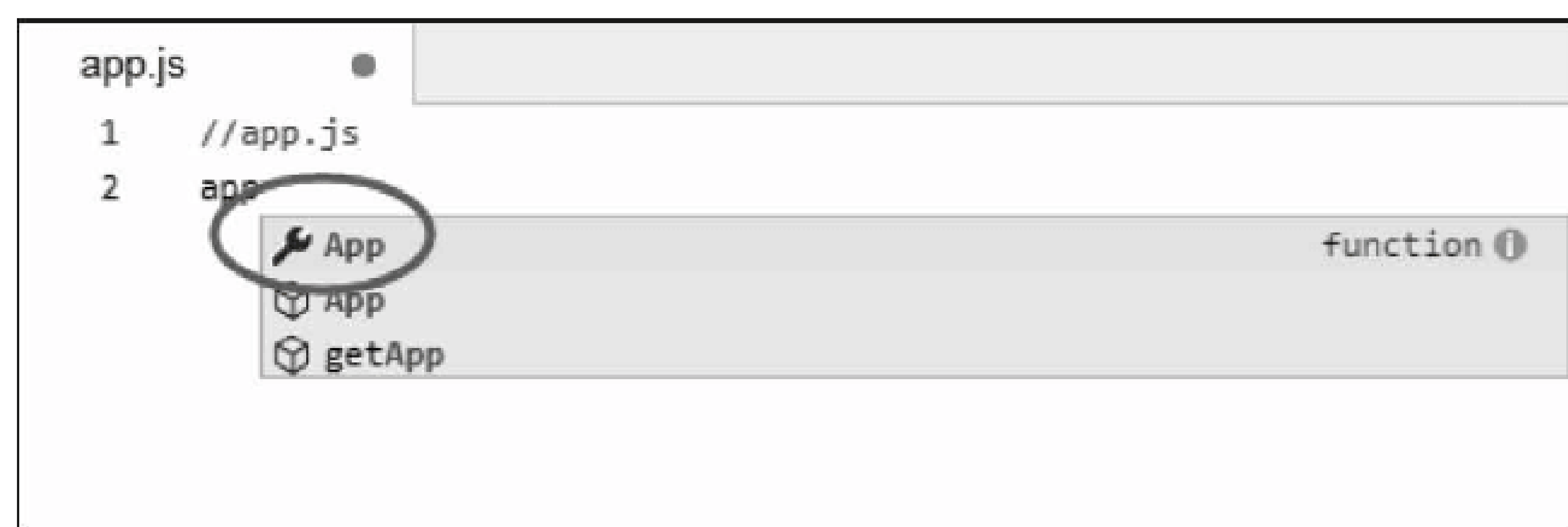


图2-10 输入关键词创建App函数

此时模板代码就修改完毕了,如果未来开发者还想创建其他页面,只需在app.json的pages属性中追加声明路径即可自动生成对应的页面文件。

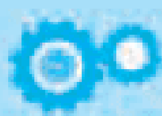
2.2.3 视图设计



视频讲解

1 导航栏设计

小程序默认导航栏是黑底白字的效果,可以通过在app.json中对window属性进行重新配置来自定义导航栏效果。更改后的app.json文件代

码如下：

```
1.  {
2.    "pages": [
3.      "pages/index/index"
4.    ],
5.    "window": {
6.      "navigationBarBackgroundColor": "#663399",
7.      "navigationBarTitleText": "手动创建第一个小程序"
8.    }
9.  }
```



图 2-11 自定义导航栏效果

上述代码可以更改导航栏背景色为紫色、字体为白色，效果如图 2-11 所示。


视频讲解

2 页面设计

页面上主要包含 3 个内容，即微信头像、微信昵称和“点击获取头像和昵称”按钮。页面设计图如图 2-12 所示。

计划使用如下组件。

- 微信头像：<image>(图像)组件；
- 微信昵称：<text>(文本)组件；
- 按钮：<button>(按钮)组件。

相关 WXML(pages/index/index.wxml)代码片段如下：

```
1.  <view class = 'container'>
2.    <image></image>
3.    <text>Hello World</text>
4.    <button>点击获取头像和昵称</button>
5.  </view>
```

相关 WXSS(pages/index/index.wxss)代码片段如下：

```
1.  .container{
2.    height: 100vh;                        /* 高 100 视窗,这里写 100% 无效 */
3.    display: flex;                        /* flex 布局模式 */
4.    flex-direction: column;               /* 垂直布局 */
5.    align-items: center;                  /* 水平方向居中 */
6.    justify-content: space-around;        /* 垂直方向分散布局 */
7.  }
```

当前效果如图 2-13 所示。

由图可见，此时可以显示文本和按钮。由于尚未获得头像图片，所以无法显示内容。可以临时使用本地图片代替，在点击按钮获取微信头像后再更改。

在项目中新建自定义文件夹 images 用于存放图片，右击此文件夹，选择“硬盘打开”，将本地图片 logo.png 复制、粘贴进去等待使用。新目录结构如图 2-14 所示。

修改 WXML 页面的<image>组件如下：

```
<image src = '/images/logo.png' mode = 'widthFix'></image>
```

上述代码中 src 属性用于指定图片来源为根目录下 images 文件夹中的 logo.png 图片，mode 属性表示图片随着指定的宽度自动拉伸高度以显示原图的正确比例。

在 WXSS 页面追加<image>和<text>组件的相关样式代码如下：

```
1.  image {
2.    width: 300rpx;                      /* 图片宽度 */
3.    border-radius: 50%;                 /* 4 个角变为圆角形状 */
4.  }
5.  text{
6.    font-size: 50rpx;                   /* 字体大小 */
7.  }
```

当前页面效果如图 2-15 所示。

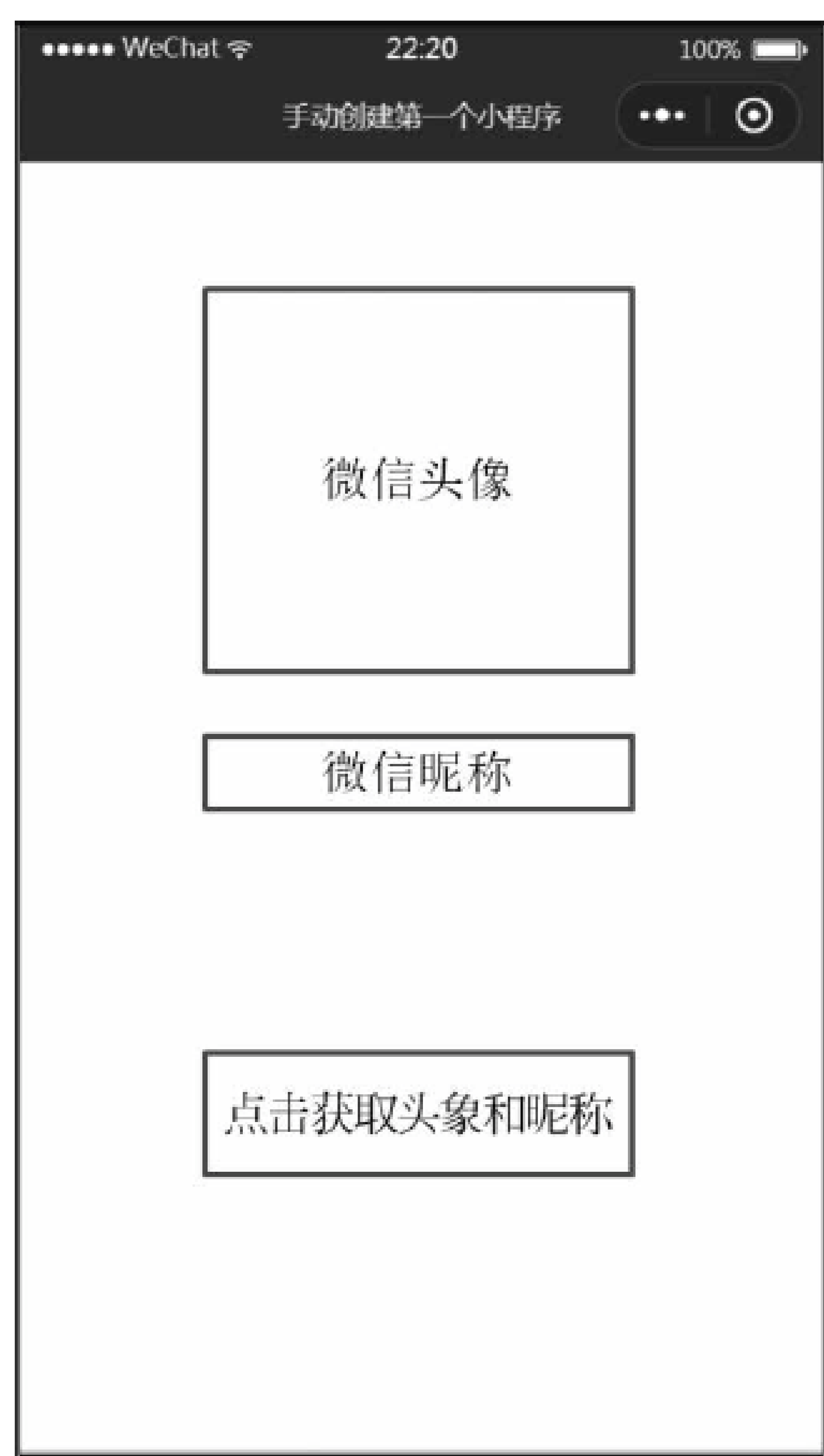


图 2-12　页面设计图

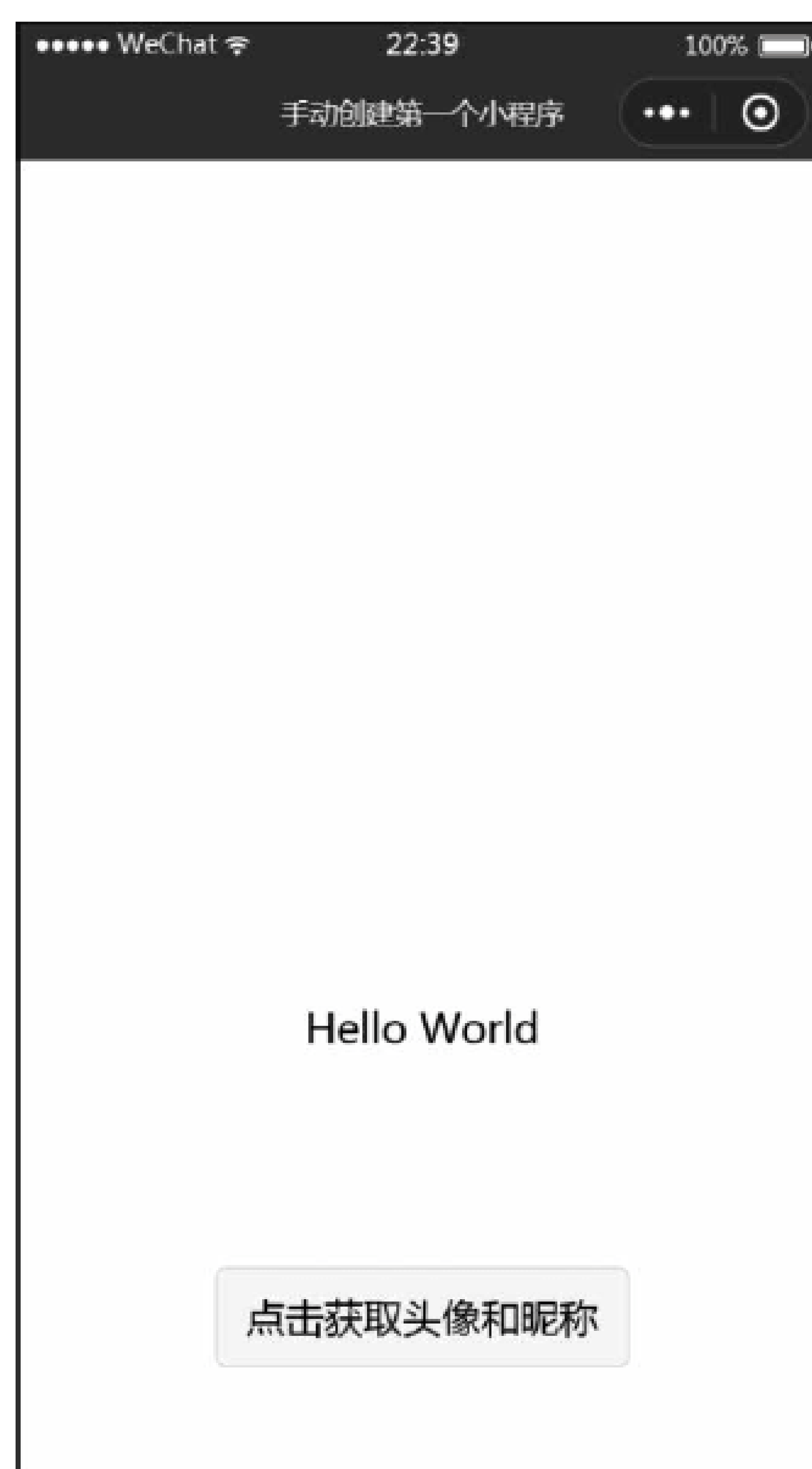


图 2-13　页面设计图

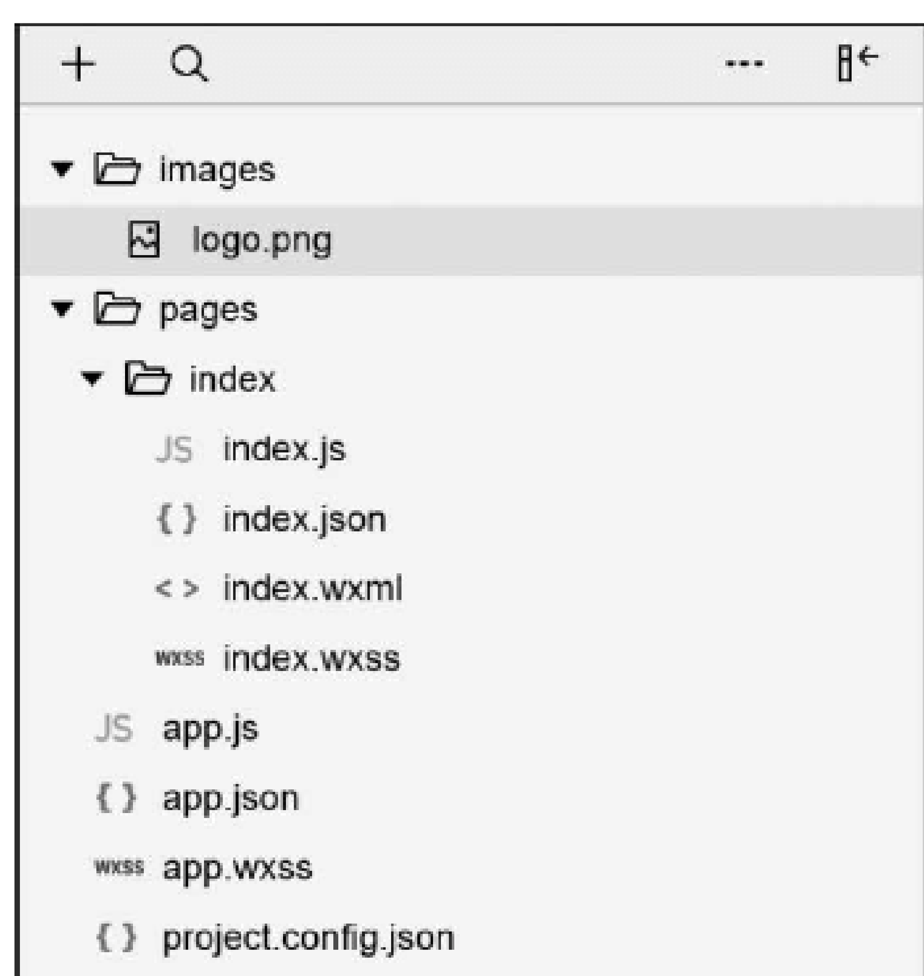


图 2-14　添加图片文件后的目录结构



图 2-15　页面效果图



此时页面布局与样式设计就已完成，2.2.4 节将介绍如何点击按钮获取头像和昵称数据。

2.2.4 逻辑实现



视频讲解

1 获取微信用户信息

在 WXML 页面修改<button>组件的代码，为其追加获取用户信息事件，代码片段如下：

```
1.  <button open-type='getUserInfo' bindgetuserinfo='getMyInfo'>
2.      点击获取头像和昵称
3.  </button>
```

其中 open-type= getUserInfo'表示激活获取微信用户信息功能，然后使用 bindgetuserinfo 属性表示获得的数据将传递给自定义函数 getMyInfo，开发者也可以使用其他名称。

在 JS 页面的 Page()内部追加 getMyInfo 函数，代码片段如下：

```
1.  getMyInfo: function(e) {
2.      console.log(e.detail.userInfo)
3.  },
```

保存后预览项目，如果点击按钮后 Console 控制台能够成功输出一段数据，就说明获取成功，如图 2-16 所示。

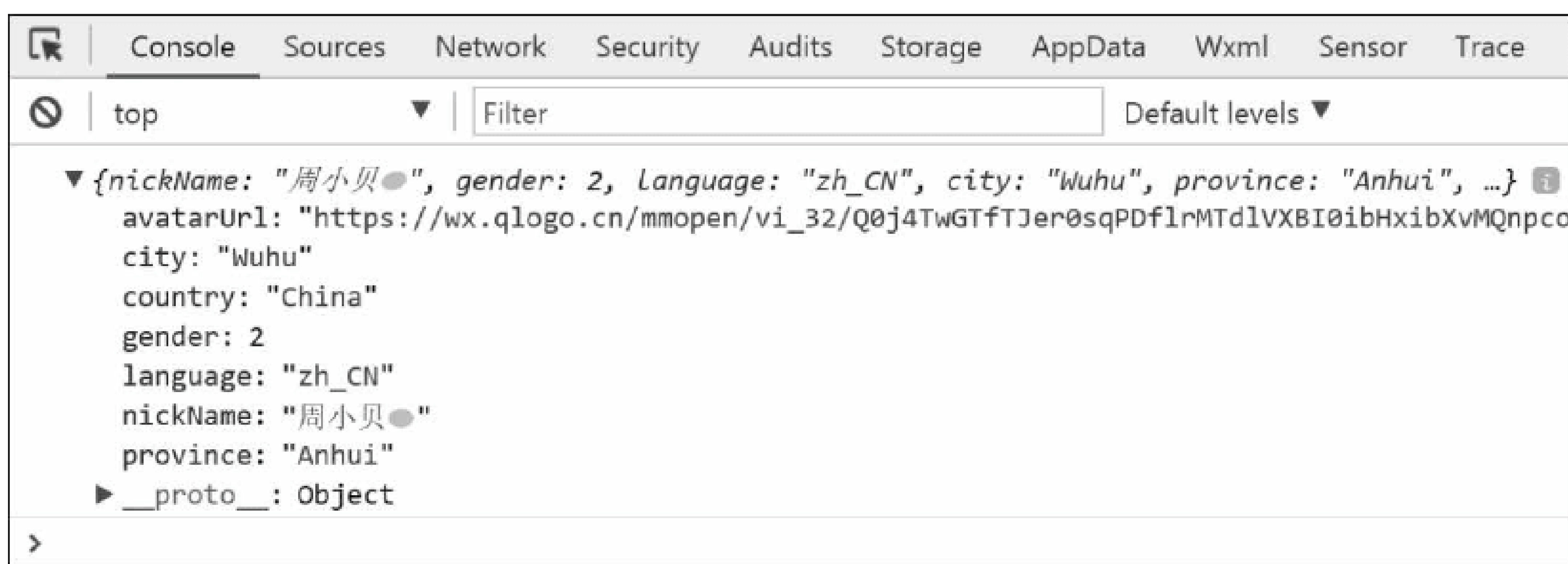


图 2-16 Console 控制台输出内容

2 使用动态数据显示头像和昵称



视频讲解

在 WXML 页面修改<image>和<text>组件的代码，将图片来源和文本内容使用双花括号（{{}}）做成动态数据，代码片段如下：

```
1.  <image src='{{src}}' mode='widthFix'></image>
2.  <text>{{name}}</text>
```

同时修改 JS 文件，在现有的 data 属性中追加这两个动态数据的值，使其仍然可以显示原先的内容。代码片段如下：

```
1.  data: {
2.    src:'/images/logo.png',
3.    name:'Hello World'
4.  },
```

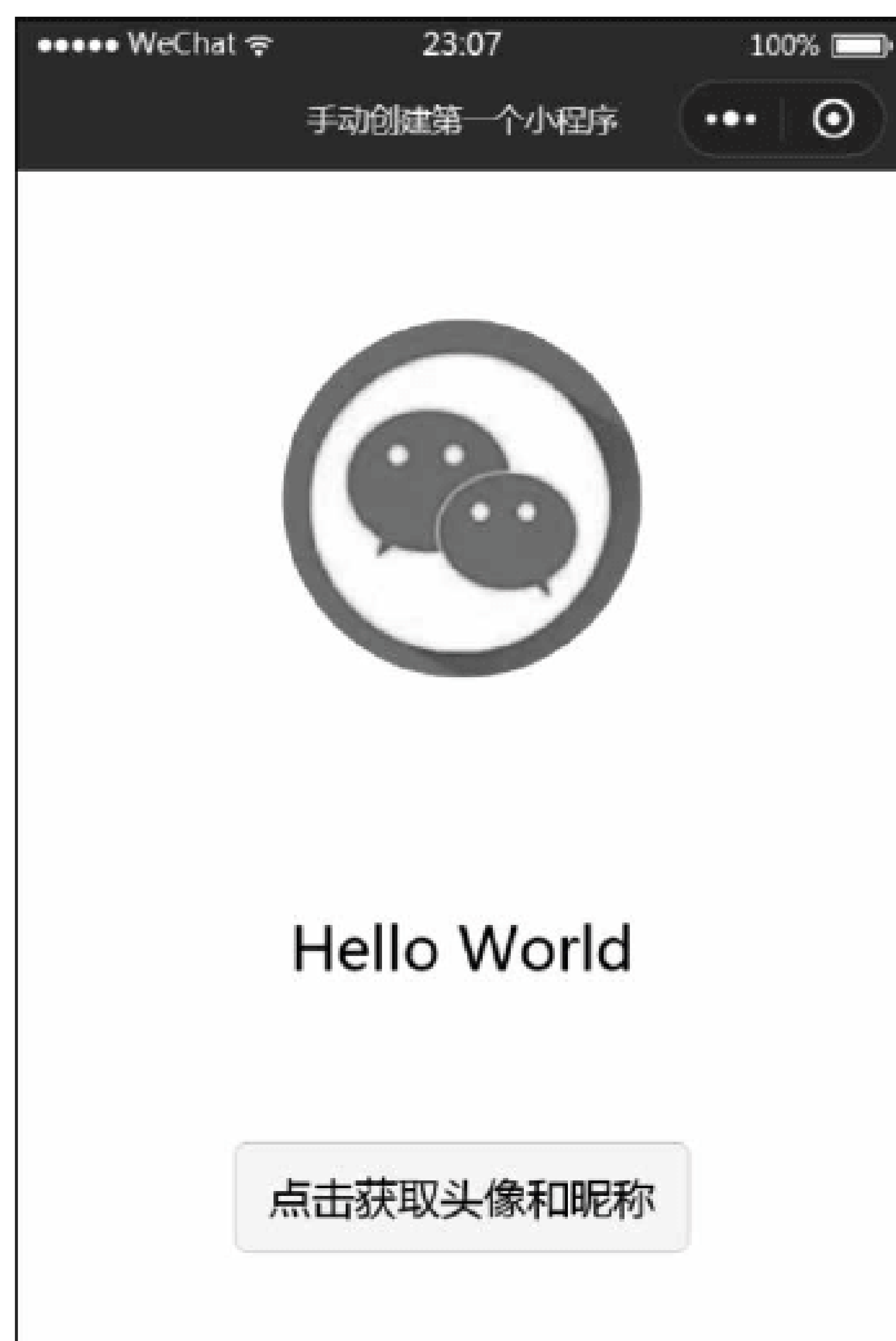


图 2-17 页面效果图

完成后预览如图 2-17 所示，页面显示效果不会发生改变。



视频讲解

3 更新头像和昵称

修改 JS 文件中 getMyInfo 函数的代码，使获取到的信息更新到动态数据上，代码片段如下：

```
1.    getMyInfo: function(e) {
2.      let info = e.detail.userInfo;
3.      this.setData({
4.        src: info.avatarUrl,                    //更新图片来源
5.        name: info.nickName                     //更新昵称
6.      })
7.    },
```

此时就已全部完成，保存后重新预览项目，最终效果图如图 2-18 所示。



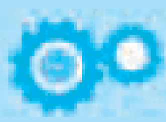
(a) 初始页面效果　　(b) 点击按钮后的效果

图 2-18　最终效果图

2.2.5　完整代码展示

app.json 文件的完整代码如下：

```
1.  {
2.    "pages": [
3.      "pages/index/index"
4.    ],
5.    "window": {
6.      "navigationBarBackgroundColor": "#663399",
7.      "navigationBarTitleText": "手动创建第一个小程序",
8.      "navigationBarTextStyle": "white"
9.    }
10. }
```

WXML 文件(pages/index/index.wxml)的完整代码如下：

```
1.  <view class = 'container'>
2.    <image src = '{{src}}' mode = 'widthFix'></image>
3.    <text>{{name}}</text>
4.    <button open-type = 'getUserInfo' bindgetuserinfo = 'getMyInfo'>
5.        点击获取头像和昵称
6.    </button>
7.  </view>
```

WXSS 文件(pages/index/index.wxss)的完整代码如下：

```
1.  .container {
2.    height: 100vh;                          /* 高 100 视窗,这里写 100% 无效 */
3.    display: flex;                          /* flex 布局模式 */
4.    flex-direction: column;                 /* 垂直布局 */
5.    align-items: center;                    /* 水平方向居中 */
6.    justify-content: space-around;          /* 垂直方向分散布局 */
7.  }
8.  image {
9.    width: 300rpx;                          /* 图片宽度 */
10.   border-radius: 50%;                     /* 4 个角变为圆角形状 */
11. }
12. text {
13.   font-size: 50rpx;                       /* 字体大小 */
14. }
```

JS 文件(pages/index/index.js)的完整代码如下：

```
1.  Page({
2.    /**
3.     * 页面的初始数据
4.     */
5.    data: {
6.      src: '/images/logo.png',
7.      name: 'Hello World'
8.    },
9.    /**
10.    * 自定义函数--获取微信用户信息
11.    */
12.   getMyInfo: function(e) {
13.     let info = e.detail.userInfo;
14.     this.setData({
15.       src: info.avatarUrl,                //更新图片来源
16.       name: info.nickName                 //更新昵称
17.     })
18.   }
19. })
```

基础篇



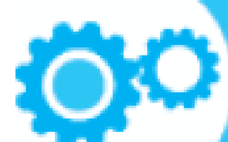
第3章 Chapter 3

小程序框架

本章内容主要包含两个使用 flex 布局模型创建的小程序项目实例，一是仿微信“发现”页面创建列表布局小程序；二是仿微信“钱包”页面创建九宫格布局小程序。

本章学习目标

- 学习使用 flex 布局模型和 wx:for 属性创建列表布局小程序；
- 学习使用 flex 布局模型和 wx:for 属性创建九宫格布局小程序。



3.1 列表布局小程序



微信 App 的“发现”页面是由若干个垂直排列的列表组成的，每个列表项均包含图标、文字和箭头符号，如图 3-1 所示。

本项目将使用 flex 布局模型和 wx:for 属性仿微信“发现”页面实现列表布局效果。



图 3-1 微信 App“发现”页面真机截屏

3.1.1 项目创建



视频讲解

本项目创建选择空白文件夹 wxDiscover，效果如图 3-2 所示。

单击“新建”按钮完成项目创建，然后准备手动创建页面配置文件。

3.1.2 页面配置



视频讲解

1 创建页面文件

项目创建完毕后，在根目录中会生成文件夹 pages 用于存放页面文件。一般来说首页默认命名为 index，表示小程序运行的第一个页面；其他页面名称可以自定义。本项目只需要保留首页(index)即可。

具体操作如下：

(1) 将 app.json 文件内 pages 属性中的“pages/logs/logs”删除，并删除上一行末尾的逗号。

(2) 按快捷键 Ctrl+S 保存当前修改。

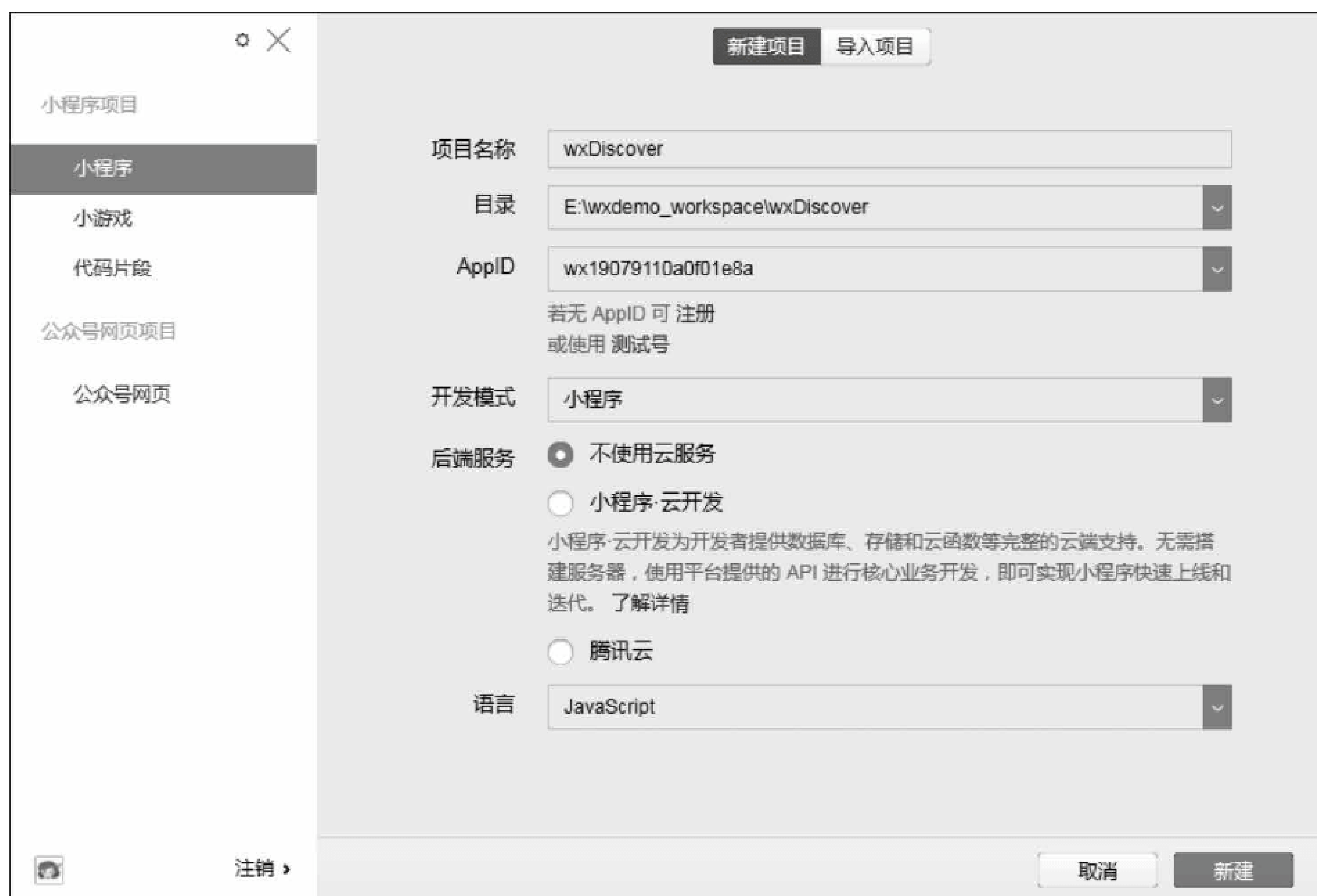


图 3-2　小程序项目填写效果示意图

2 删除和修改文件

具体操作如下：

（1）删除 utils 文件夹及其内部所有内容。

（2）删除 pages 文件夹下的 logs 目录及其内部所有内容。

（3）删除 index. wxml 和 index. wxss 中的全部代码。

（4）删除 index. js 中的全部代码，并且输入关键词 page 找到第二个选项按回车键让其自动补全函数（如图 3-3 所示）。

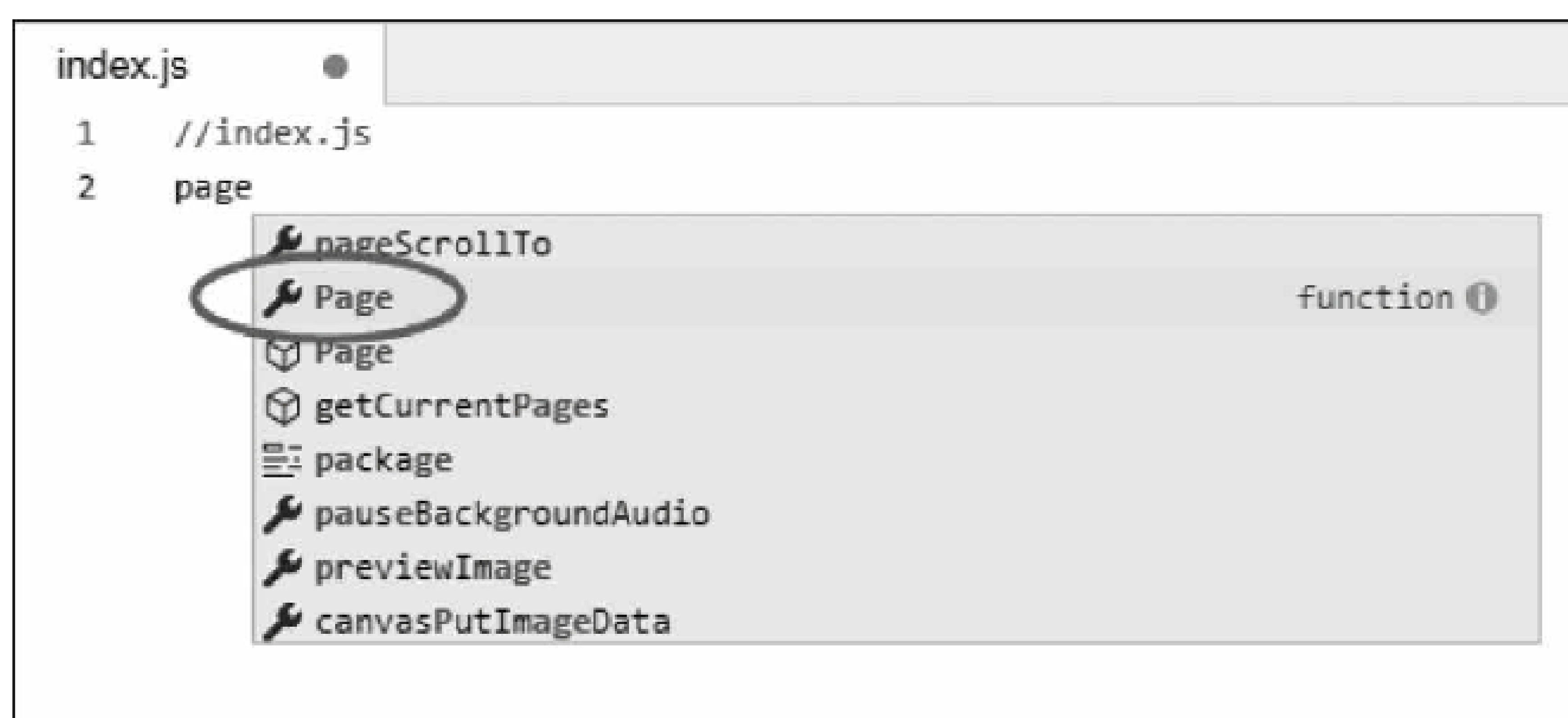


图 3-3　输入关键词创建 Page 函数

（5）删除 app. wxss 中的全部代码。

（6）删除 app. js 中的全部代码，并且输入关键词 app 找到第一个选项按回车键让其自动补全函数（如图 3-4 所示）。

3 创建其他文件

接下来创建其他自定义文件，本项目还需要一个文件夹用于存放图标素材。文件夹名称由开发者自定义（例如 images），单击目录结构左上角的＋号创建文件夹并命名为 images。

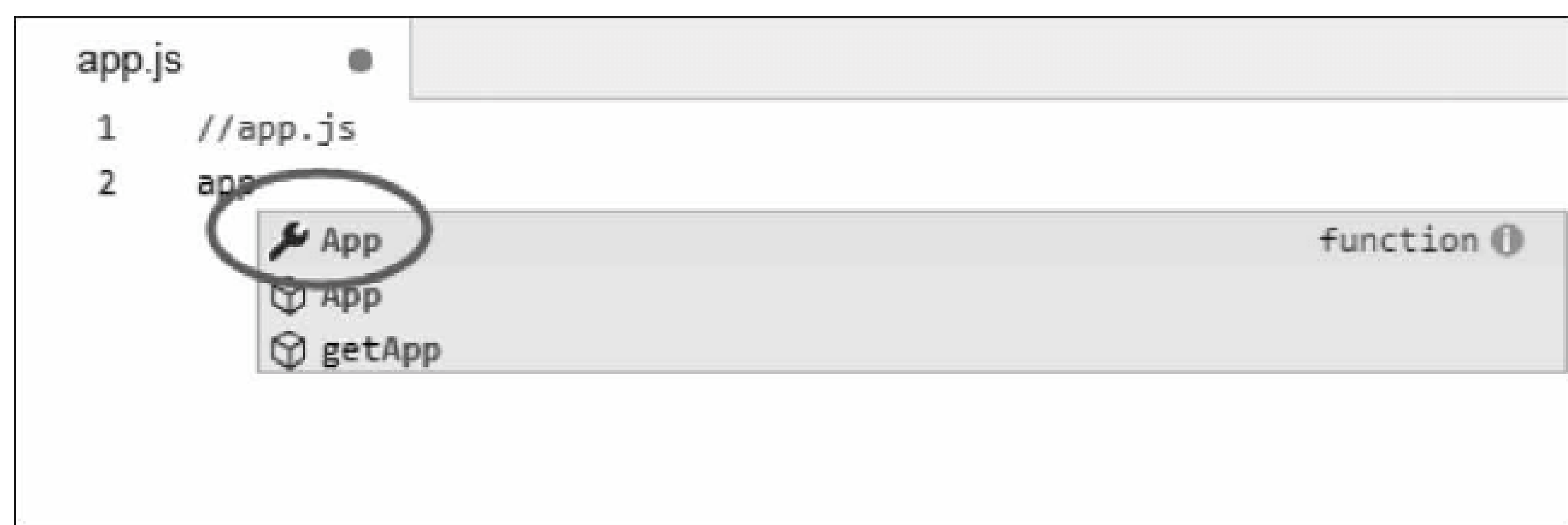


图 3-4 输入关键词创建 App 函数

本项目将用到 8 个列表项图标和 1 个通用箭头图标，图片素材如图 3-5 所示。

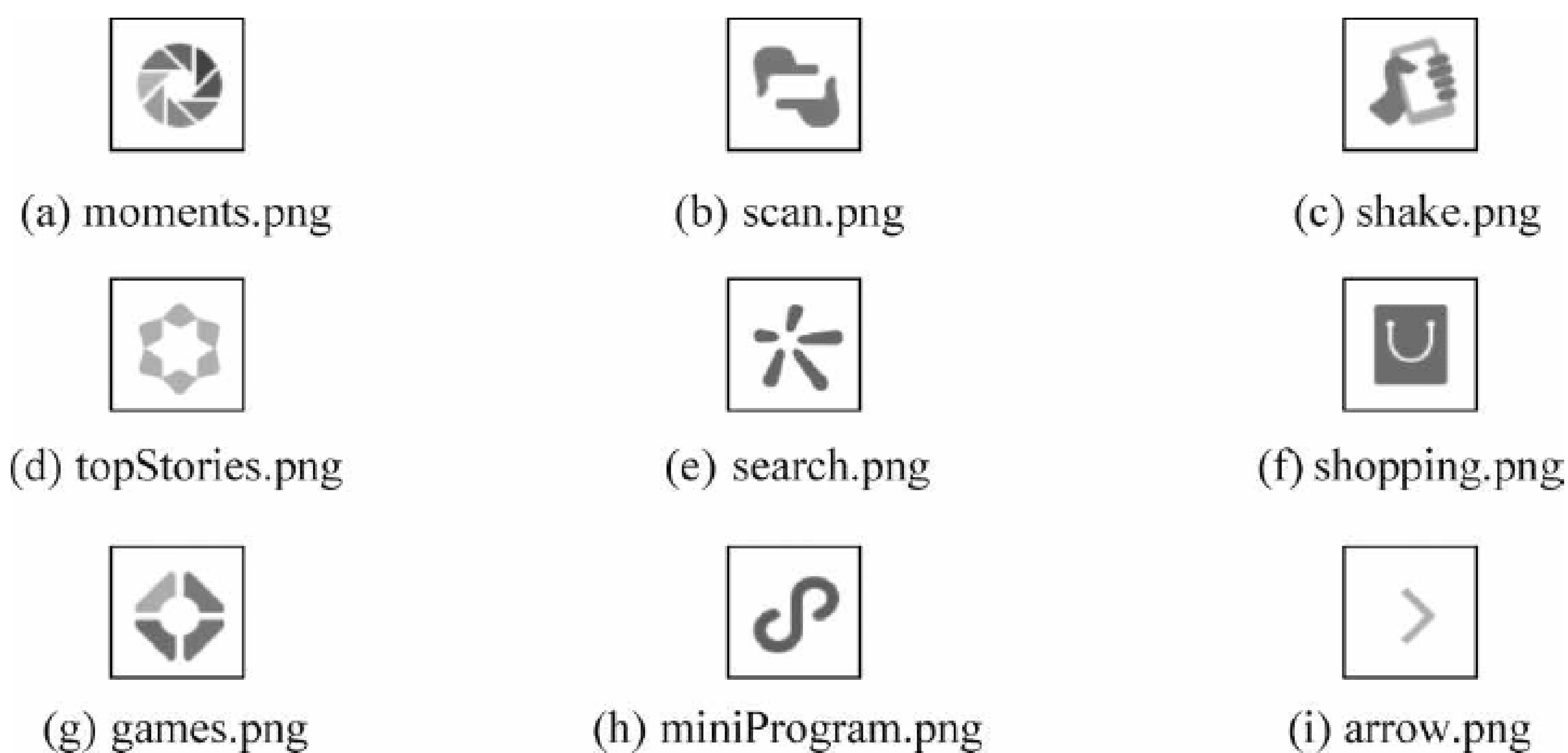


图 3-5 图标素材展示

右击目录结构中的 images 文件夹，选择“硬盘打开”，将图片复制、粘贴进去。

全部完成后的目录结构如图 3-6 所示。

此时文件配置就全部完成，3.1.3 节将正式进行页面布局和样式设计。

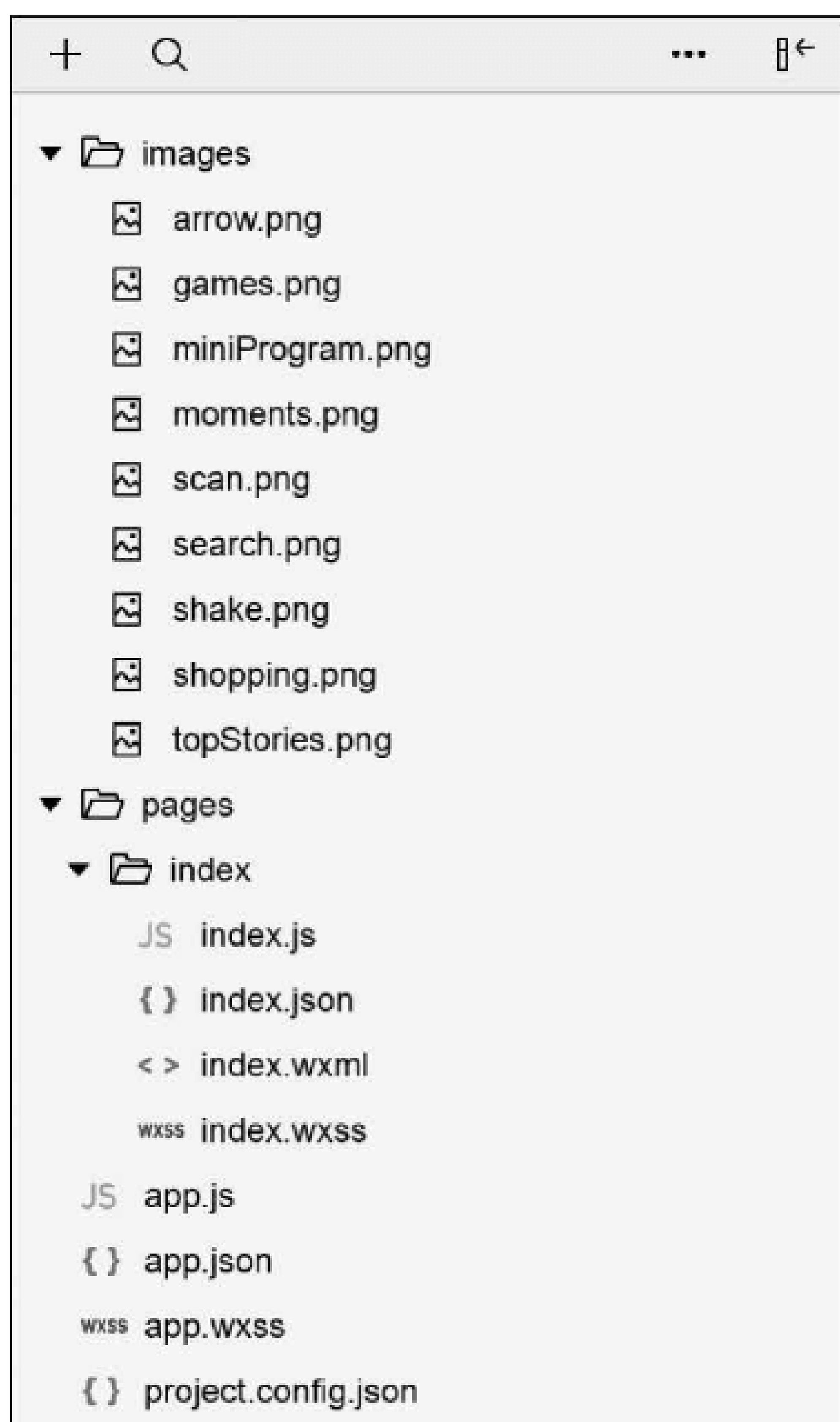


图 3-6 页面文件创建完成

3.1.3 视图设计



视频讲解

1 导航栏设计

小程序默认导航栏就是黑底白字的效果，因此只需要在 index.json 中自定义导航栏标题即可。更改后的 index.json 文件代码如下：

```
1.  {
2.      "navigationBarTitleText": "发现"
3.  }
```

上述代码可以更改当前页面的导航栏标题文本为“发现”，效果如图 3-7 所示。

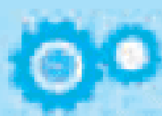


视频讲解

2 页面设计

页面上主要包含 5 组列表，每组列表包含 1～2 个列表项，具体内容解释如下。

- 列表组 1：“朋友圈”单行列表项；
- 列表组 2：“扫一扫”和“摇一摇”两行列表项；
- 列表组 3：“看一看”和“搜一搜”两行列表项；



• 列表组 4："购物"和"游戏"两行列表项；



图 3-7　自定义导航栏效果

• 列表组 5："小程序"单行列表项。

每个列表组之间需要有一定的间隔距离，设计图如图 3-8 所示。

计划使用如下组件。

• 页面整体：<view>组件，并定义 class='container'；
• 列表组：<view>组件，并定义 class='listGroup'；
• 列表项单行：<view>组件，并定义 class='listItem'；
• 列表图标：<image>(图像)组件；
• 列表文字内容：<text>(文本)组件；
• 箭头图标：<image>(图像)组件。

首先定义页面容器(<view>)，WXML(pages/index/index.wxml)代码片段如下：

```
1.  <view class = 'container'>
2.  </view>
```

WXSS(pages/index/index.wxss)代码片段如下：

```
1.  /* 背景容器样式 */
2.  .container{
3.    height: 100vh;                    /* 高度为 100 视窗，写成 100% 无效 */
4.    background-color: silver;         /* 背景颜色为银色 */
5.    display: flex;                    /* flex 布局模型 */
6.    flex-direction: column;           /* 垂直布局 */
7.  }
```

当前效果如图 3-9 所示。



图 3-8　页面设计图

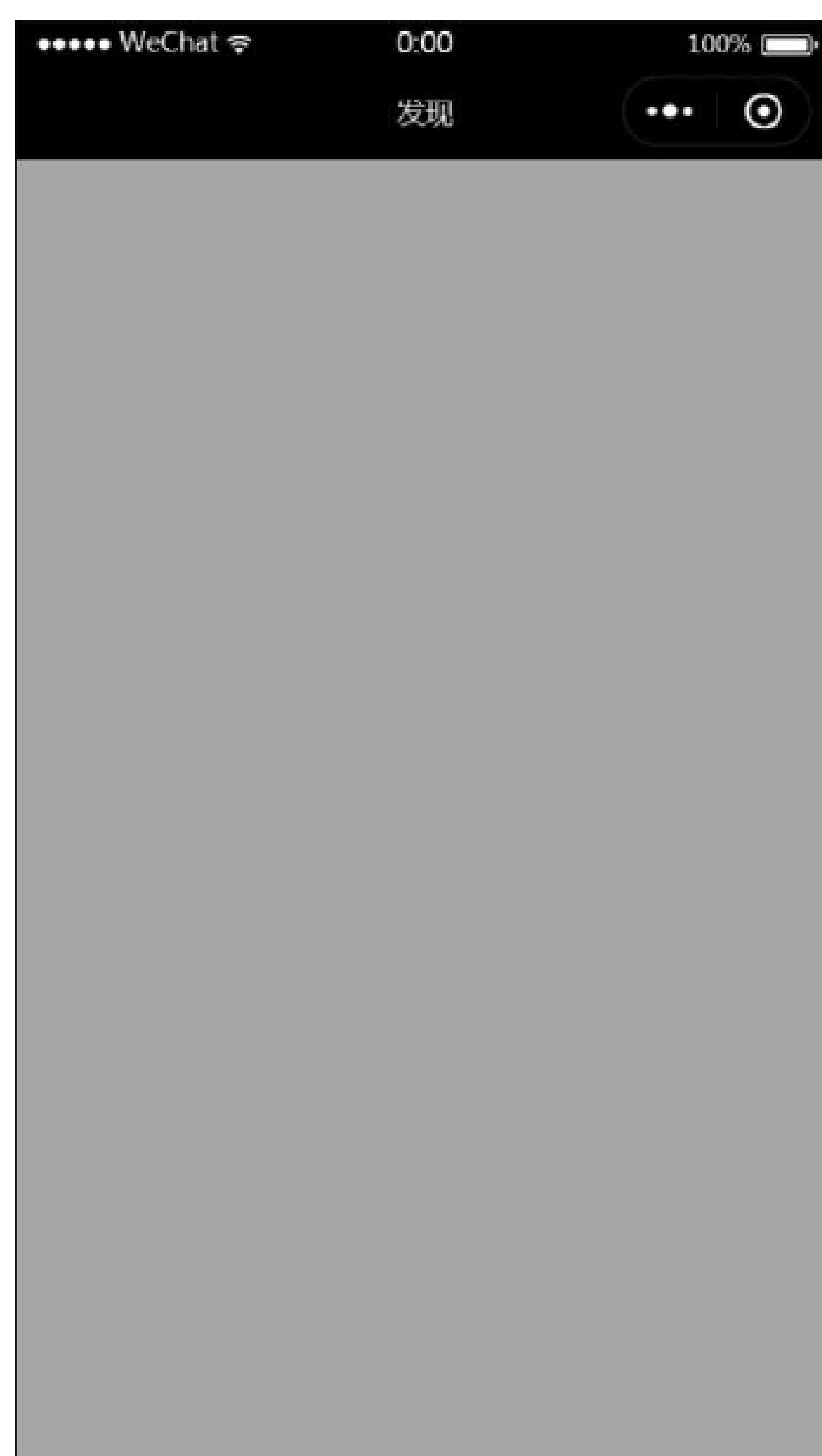


图 3-9　当前页面预览效果

由图可见，此时整个页面背景变成了银色。由于还没添加组件元素，所以尚看不出来 flex 布局模型效果。

接下来以第一个列表选项为例，WXML(pages/index/index.wxml)代码片段修改如下：

```
1.  <view class='container'>
2.    <view class='listGroup'>
3.      <view class='listItem'>
4.        <image src='/images/moments.png'></image>
5.        <text>朋友圈</text>
6.        <image src='/images/arrow.png'></image>
7.      </view>
8.    </view>
9.  </view>
```

WXSS(pages/index/index.wxss)代码片段如下：

```
1.  /* 列表组样式 */
2.  .listGroup{
3.    background-color: white;          /* 背景颜色为白色 */
4.    margin: 20rpx 0;                  /* 上下外边距 20rpx,左右 0 */
5.  }
6.  /* 列表项单行样式 */
7.  .listItem{
8.    display: flex;                    /* flex 布局模型 */
9.    flex-direction: row;              /* 水平布局 */
10.   align-items: center;              /* 垂直方向居中 */
11.   border: 1rpx solid silver;        /* 1rpx 宽的银色实线边框 */
12.   padding: 10rpx;                   /* 内边距 10rpx */
13. }
14. /* 图标的尺寸 */
15. image{
16.   width: 80rpx;                     /* 宽度 */
17.   height: 80rpx;                    /* 高度 */
18.   margin: 0 15rpx;                  /* 上下外边距 0,左右外边距 15rpx */
19. }
20. /* 文本样式 */
21. text{
22.   font-size: 40rpx;          /* 字体大小 40rpx */
23.   flex-grow: 1;              /* 扩张多余空间宽度 */
24. }
```

当前效果如图 3-10 所示。

由图可见，此时可以显示第一个列表组的内容。用同样的方式追加后续的列表组即可实现完整效果。当然也可以暂时不追加其他列表项，使用 3.1.4 节介绍的方法减少工作量。

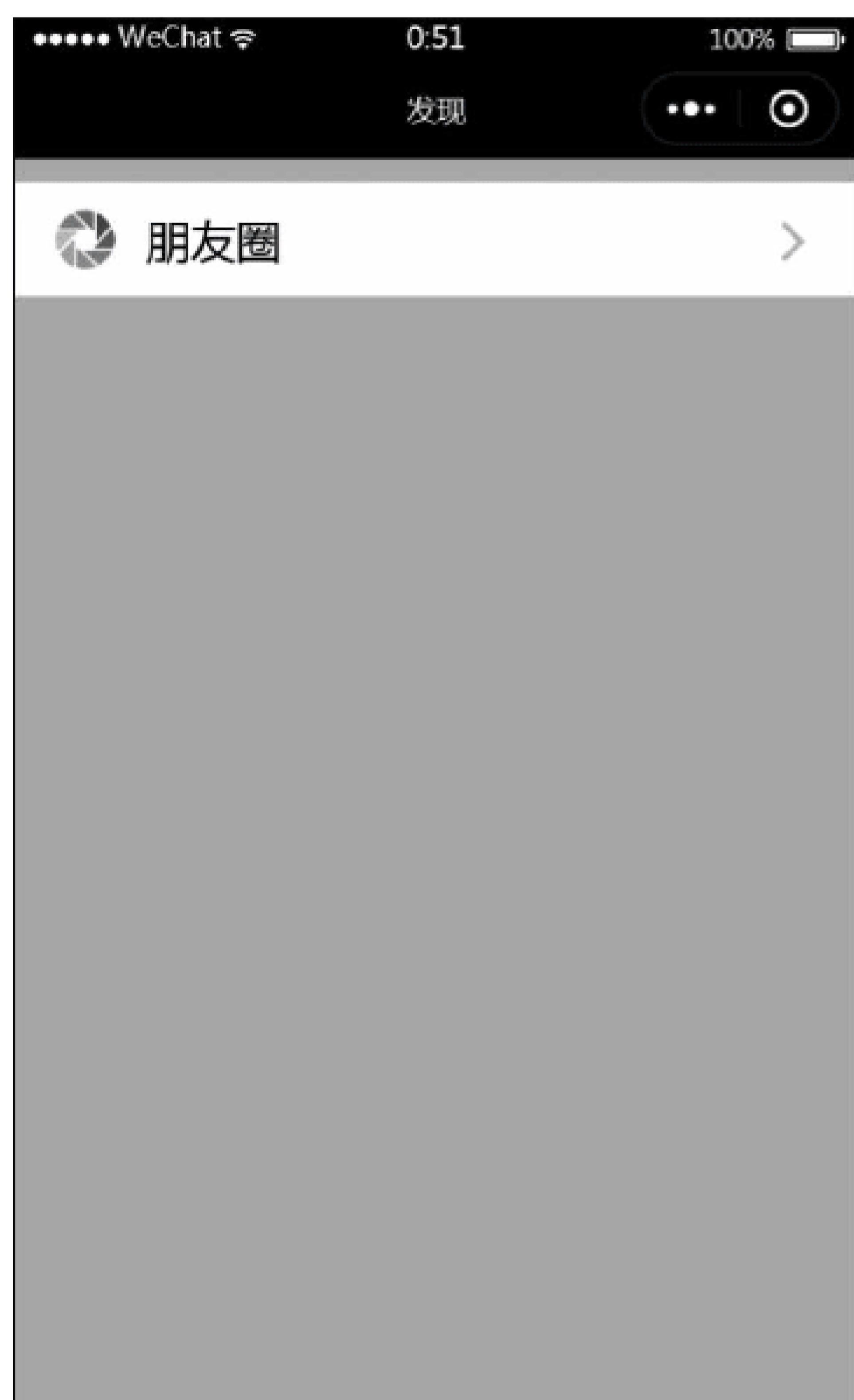


图 3-10 当前页面预览效果

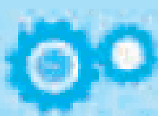
3.1.4 逻辑实现



视频讲解

1 使用动态数据展示列表

由于所有列表项的内容布局都是统一的，可以考虑使用 wx:for 属性配合动态数组渲染全部列表项，以减少 WXML 页面的代码量。



修改 WXML(pages/index/index.wxml)页面代码如下：

```
1.  <view class='container'>
2.    <view class='listGroup' wx:for='{{list}}' wx:for-item='group' wx:key='group{{index}}'>
3.      <view class='listItem' wx:for='{{group}}' wx:for-item='row' wx:key='row{{index}}'>
4.        <image class='icon' src='{{row.icon}}'></image>
5.        <text>{{row.text}}</text>
6.        <image src='/images/arrow.png'></image>
7.      </view>
8.    </view>
9.  </view>
```

上述代码表示将使用双重 wx:for 属性循环显示全部列表项，其中{{list}}数组用于表示 5 个列表组，并为每个列表组起了别名 group；每个列表项也起了别名 row，列表项的图标和文本分别命名为 icon、text。这里均为自定义名称，开发者可以自行更改。



视频讲解

2 补充数组完整信息

在 index.js 的 data 属性中添加 list 数组，JS 文件(pages/index/index.js)代码如下：

```
1.  Page({
2.    data: {
3.      list: [
4.        //第 1 组列表
5.        [{ text: '朋友圈', icon: '/images/moments.png' }],
6.        //第 2 组列表
7.        [
8.          { text: '扫一扫', icon: '/images/scan.png' },
9.          { text: '摇一摇', icon: '/images/shake.png' }
10.       ],
11.       //第 3 组列表
12.       [
13.         { text: '看一看', icon: '/images/topStories.
png' },
14.         { text: '搜一搜', icon: '/images/search.png' }
15.       ],
16.       //第 4 组列表
17.       [
18.         { text: '购物', icon: '/images/shopping.png' },
19.         { text: '游戏', icon: '/images/games.png' }
20.       ],
21.       //第 5 组列表
22.       [{ text: '小程序', icon: '/images/miniProgram.
png' }]
23.     ]
24.   }
25. })
```

此时就已全部完成，保存后重新预览项目，最终效果图如图 3-11 所示。



图 3-11　最终效果图

3.1.5　完整代码展示

app.json 文件的完整代码如下：

```
1.  {
2.    "pages": [
3.      "pages/index/index"
4.    ]
5.  }
```

JSON 文件(pages/index/index.json)的完整代码如下：

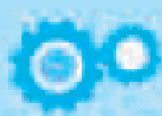
```
1.  {
2.    "navigationBarTitleText": "发现"
3.  }
```

WXML 文件(pages/index/index.wxml)的完整代码如下：

```
1.  <view class='container'>
2.    <view class='listGroup' wx:for='{{list}}' wx:for-item='group' wx:key='group{{index}}'>
3.      <view class='listItem' wx:for='{{group}}' wx:for-item='row' wx:key='row{{index}}'>
4.        <image src='{{row.icon}}'></image>
5.        <text>{{row.text}}</text>
6.        <image src='/images/arrow.png'></image>
7.      </view>
8.    </view>
9.  </view>
```

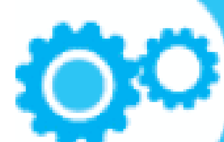
WXSS 文件(pages/index/index.wxss)的完整代码如下：

```
1.  /*背景容器样式*/
2.  .container{
3.    height: 100vh;                       /*高度为100视窗,写成100%无效*/
4.    background-color: silver;            /*背景颜色为银色*/
5.    display: flex;                       /*flex布局模型*/
6.    flex-direction: column;              /*垂直布局*/
7.  }
8.  /*列表组样式*/
9.  .listGroup{
10.   background-color: white;             /*背景颜色为白色*/
11.   margin: 20rpx 0;                     /*上下外边距20rpx,左右0*/
12. }
13. /*列表项单行样式*/
14. .listItem{
15.   display: flex;                       /*flex布局模型*/
16.   flex-direction: row;                 /*水平布局*/
17.   align-items: center;                 /*垂直方向居中*/
18.   border: 1rpx solid silver;           /*1rpx宽的银色实线边框*/
19.   padding: 10rpx;                      /*内边距10rpx*/
20. }
21. /*图标的尺寸*/
22. image{
23.   width: 80rpx;                        /*宽度*/
24.   height: 80rpx;                       /*高度*/
25.   margin: 0 15rpx;                     /*上下外边距0,左右外边距15rpx*/
26. }
27. /*文本样式*/
28. text{
29.   font-size: 40rpx;                    /*字体大小40rpx*/
30.   flex-grow: 1;                        /*扩张多余空间宽度*/
31. }
```



JS 文件(pages/index/index.js)的完整代码如下：

```
1.  Page({
2.    data: {
3.      list: [
4.        //第 1 组列表
5.        [{ text: '朋友圈', icon: '/images/moments.png' }],
6.        //第 2 组列表
7.        [
8.          { text: '扫一扫', icon: '/images/scan.png' },
9.          { text: '摇一摇', icon: '/images/shake.png' }
10.       ],
11.       //第 3 组列表
12.       [
13.         { text: '看一看', icon: '/images/topStories.png' },
14.         { text: '搜一搜', icon: '/images/search.png' }
15.       ],
16.       //第 4 组列表
17.       [
18.         { text: '购物', icon: '/images/shopping.png' },
19.         { text: '游戏', icon: '/images/games.png' }
20.       ],
21.       //第 5 组列表
22.       [{ text: '小程序', icon: '/images/miniProgram.png' }]
23.     ]
24.   }
25. })
```



3.2 九宫格布局小程序



微信 App“钱包”页面主要分为上、下两个部分，上面是由“收付款”“零钱”和“银行卡”组成的钱包状态栏，下面是由九宫格组成的“腾讯服务”栏，每个格子里面包含图标和下方的文字说明，如图 3-12 所示。

本项目将使用 flex 布局模型和 wx:for 属性仿微信“钱包”页面实现九宫格布局效果。

3.2.1 项目创建



视频讲解

本项目创建选择空白文件夹 wxWallet，效果如图 3-13 所示。

单击“新建”按钮完成项目创建，然后准备手动创建页面配置文件。

3.2.2 页面配置



视频讲解

1 创建页面文件

项目创建完毕后，在根目录中会生成文件夹 pages 用于存放页面文件。一般



图 3-12 微信 App“钱包”页面真机截屏



图 3-13　小程序项目填写效果示意图

来说首页默认命名为 index，表示小程序运行的第一个页面；其他页面名称可以自定义。本项目只需要保留首页(index)即可。

具体操作如下：

(1) 将 app.json 文件内 pages 属性中的“pages/logs/logs”删除，并删除上一行末尾的逗号。

(2) 按快捷键 Ctrl+S 保存当前修改。

2 删除和修改文件

具体操作如下：

(1) 删除 utils 文件夹及其内部所有内容。

(2) 删除 pages 文件夹下的 logs 目录及其内部所有内容。

(3) 删除 index.wxml 和 index.wxss 中的全部代码。

(4) 删除 index.js 中的全部代码，并且输入关键词 page 找到第二个选项按回车键让其自动补全函数(如图 3-14 所示)。

(5) 删除 app.wxss 中的全部代码。

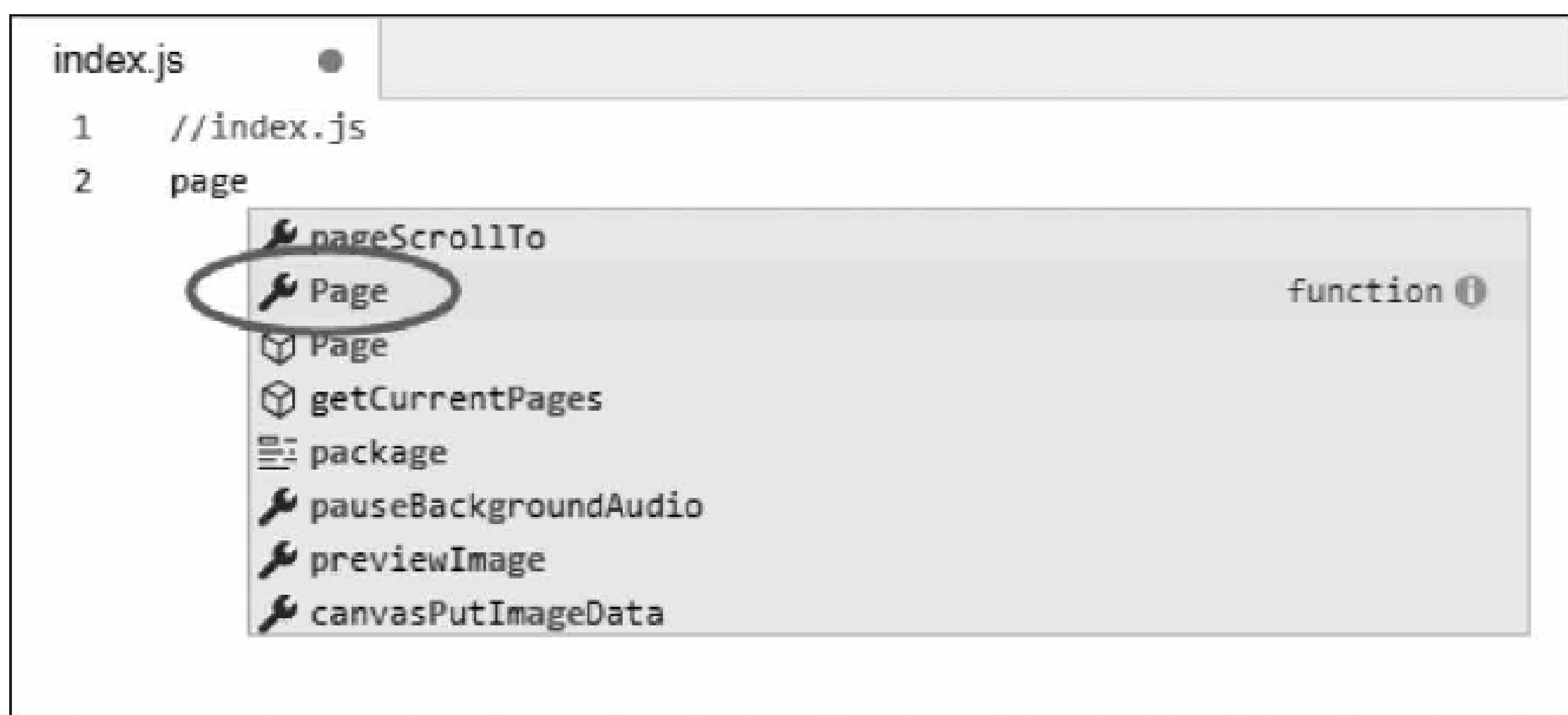
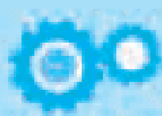


图 3-14　输入关键词创建 Page 函数



（6）删除 app.js 中的全部代码，并且输入关键词 app 找到第一个选项按回车键让其自动补全函数（如图 3-15 所示）。

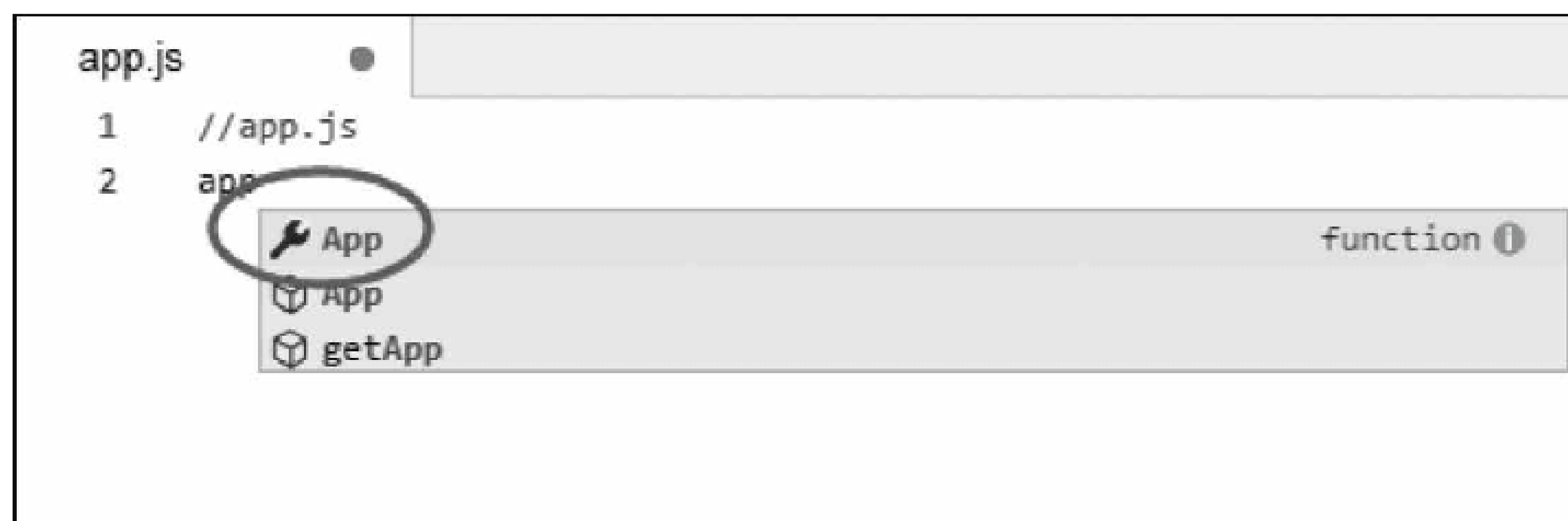


图 3-15 输入关键词创建 App 函数

3 创建其他文件

接下来创建其他自定义文件，本项目还需要一个文件夹用于存放图标素材。文件夹名称由开发者自定义（例如 images），单击目录结构左上角的＋号创建文件夹并命名为 images。

由于本项目用到的图标素材较多，将在 images 文件夹下分为两个二级目录放置。

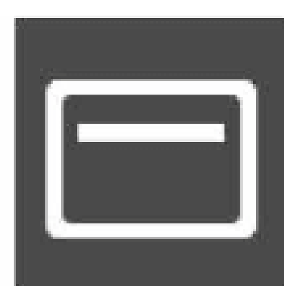
- top：顶端钱包状态栏的图标共 3 个，如图 3-16 所示。



(a) money.png



(b) balance.png



(c) cards.png

图 3-16 顶端钱包状态栏图标素材展示

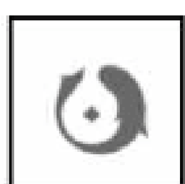
- service：“腾讯服务”栏的九宫格图标共 8 个，如图 3-17 所示。



(a) cardRepay.png



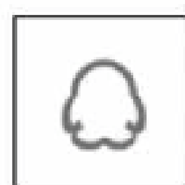
(b) mobileTopup.png



(c) wealth.png



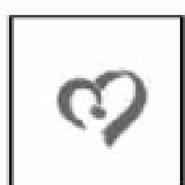
(d) utilities.png



(e) qqCoins.png



(f) publicService.png



(g) charity.png



(h) insurance.png

图 3-17 “腾讯服务”栏图标素材展示

右击目录结构中的 images 文件夹，选择“硬盘打开”，将二级目录和对应的图标文件全部复制、粘贴进去。完成后的目录结构如图 3-18 所示。

此时文件配置就全部完成，3.2.3 节将正式进行页面布局和样式设计。

3.2.3 视图设计



视频讲解

1 导航栏设计

小程序默认导航栏是黑底白字的效果，因此需要在 index.json 中自定义导航栏标题和背景颜色。更改后的 index.json 文件代码如下：

```
1. {
2.    "navigationBarTitleText": "钱包",
```

```
3.     "navigationBarBackgroundColor":"#686F79"
4.   }
```

上述代码可以更改当前页面的导航栏标题文本为“钱包”、背景颜色为灰色(#686F79)。预览效果如图 3-19 所示。

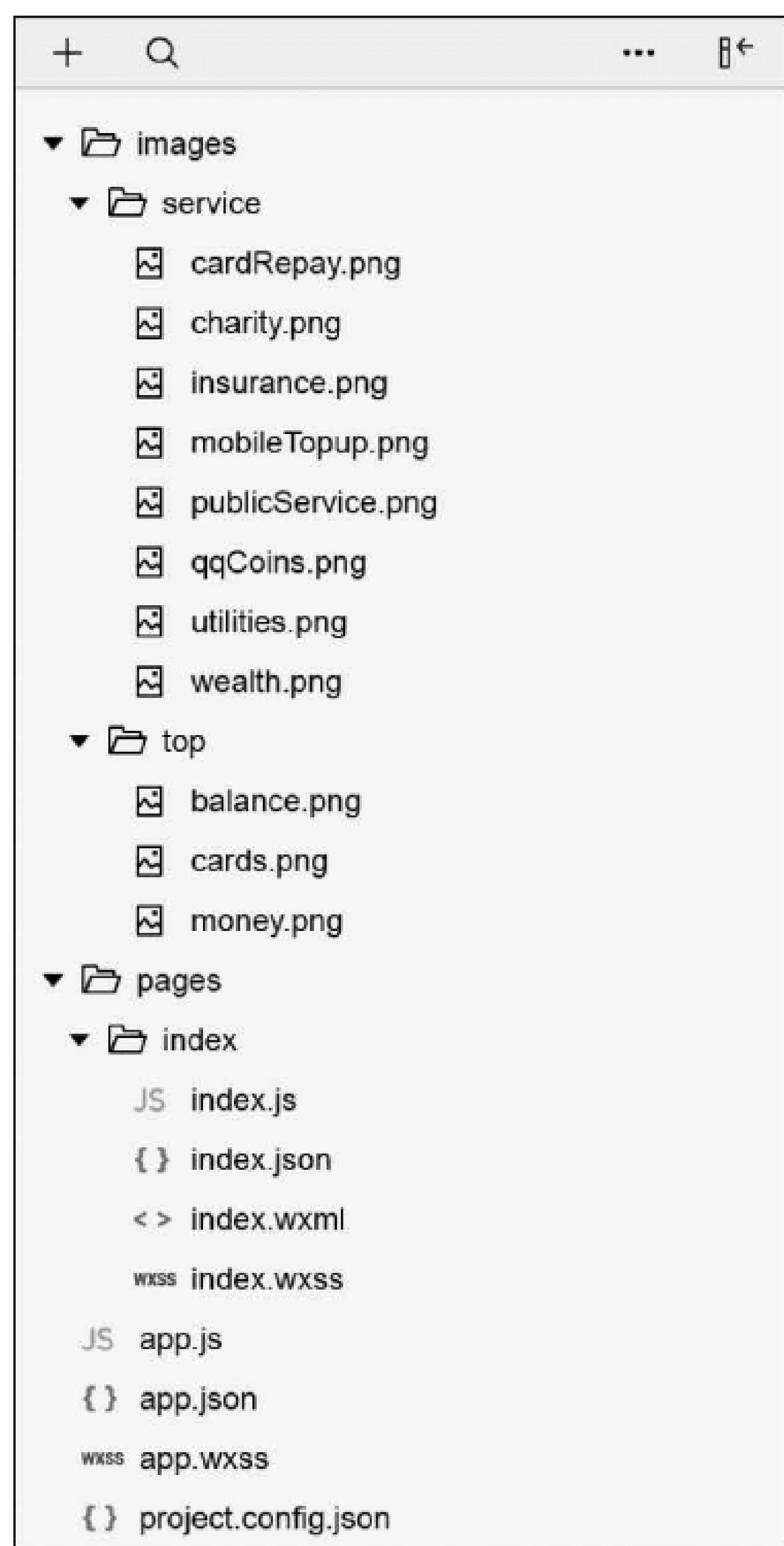


图 3-18 页面文件创建完成



图 3-19 自定义导航栏效果

2 页面设计

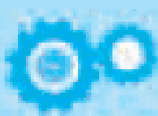
页面上主要包含两个面板，具体内容解释如下。

- 面板 1(顶端钱包状态栏)：包含“收付款”“零钱”和“银行卡”3 个方格，每个方格中均有图标、文本，其中“零钱”还包括第二行副文本。
- 面板 2(“腾讯服务”栏)：包含第一行标题和下方的九宫格区域，其中共有 8 个方格有内容，包括图标和文本。

注意，面板之间需要有一定的间隔距离，设计图如图 3-20 所示。

计划使用如下组件。

- 页面整体：<view>组件，并定义 class='container'；
- 面板 1：<view>组件，并定义 class='topPanel'；
- 面板 1 方格：<view>组件，并定义 class='box1'；
- 面板 2：<view>组件，并定义 class='servicePanel'；
- 面板 2 标题：<view>组件，并定义 class='serviceTitle'；
- 面板 2 九宫格区域：<view>组件，并定义 class='serviceBlocks'；
- 面板 2 方格：<view>组件，并定义 class='box2'；
- 方格内图标：<image>(图像)组件；



• 方格内文字内容：<text>(文本)组件。

1）整体布局设计


视频讲解

首先定义页面容器(<view>)，WXML(pages/index/index.wxml)代码片段如下：

```
1.  <view class = 'container'>
2.  </view>
```

WXSS(pages/index/index.wxss)代码片段如下：

```
1.  /* 1 背景容器样式 */
2.  .container{
3.    height: 100vh;                    /* 高度为 100 视窗，写成 100% 无效 */
4.    background-color: silver;         /* 背景颜色为银色 */
5.    display: flex;                    /* flex 布局模型 */
6.    flex-direction: column;           /* 垂直布局 */
7.  }
```

当前效果如图 3-21 所示。

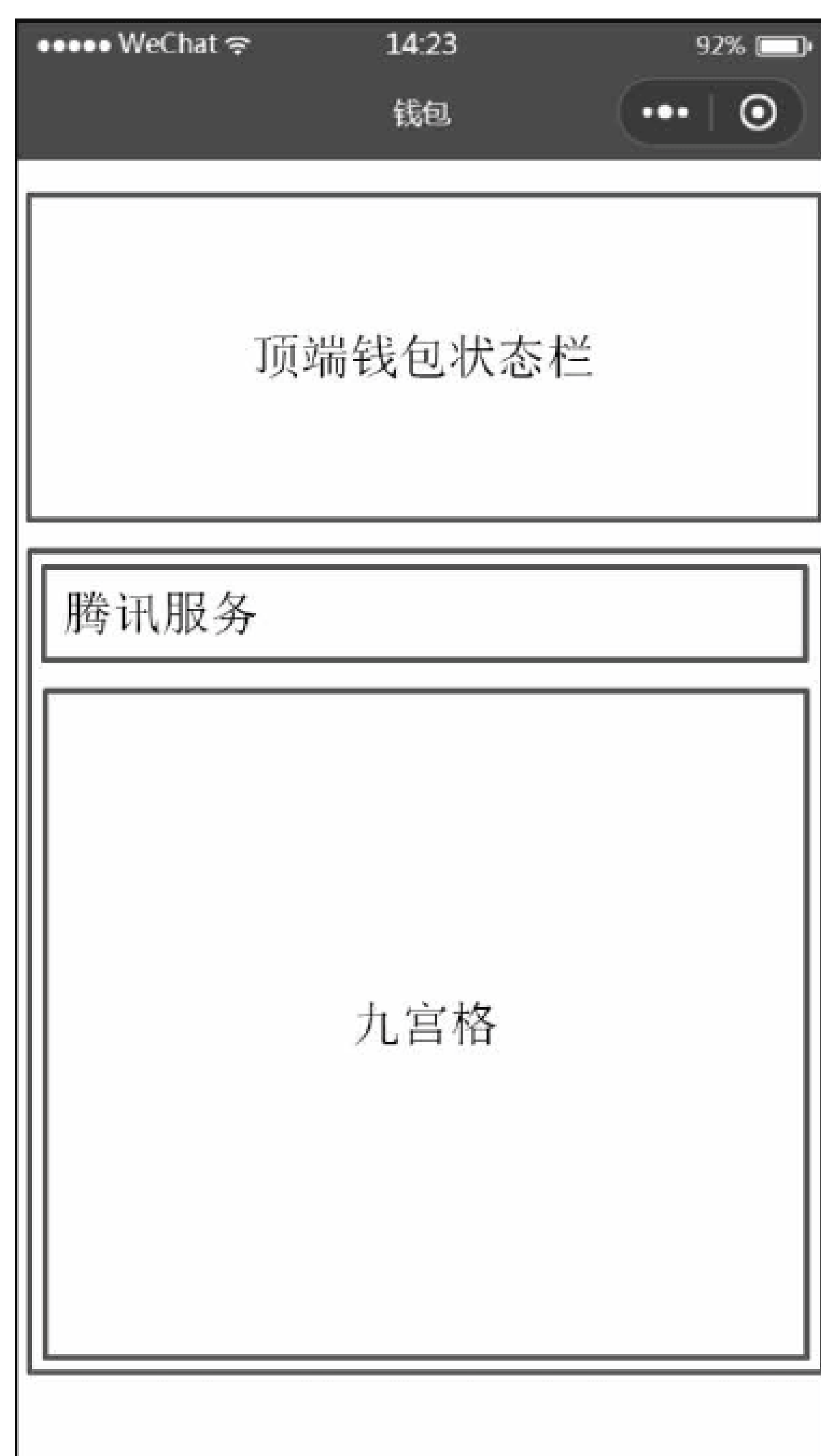


图 3-20 页面设计图

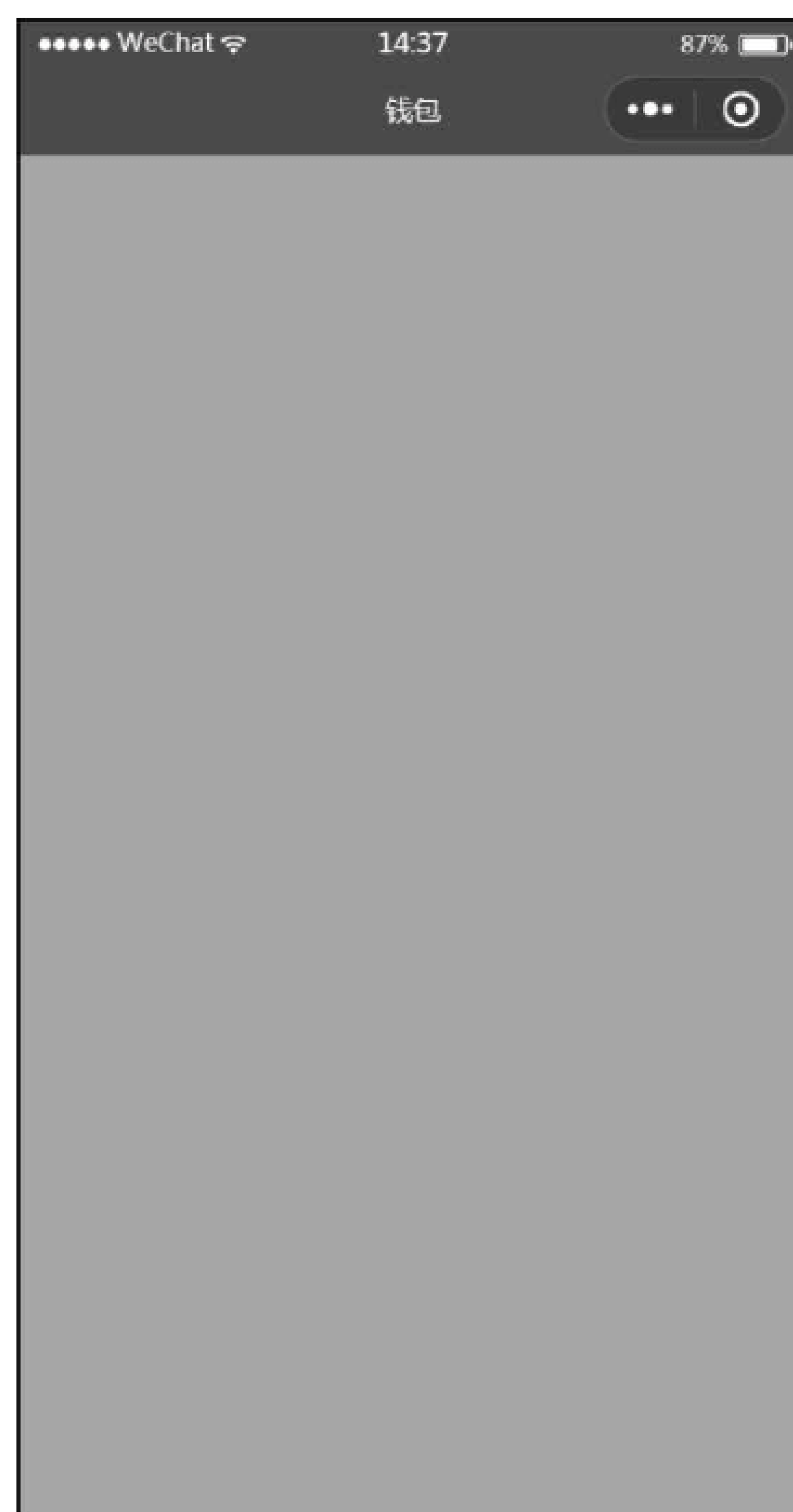


图 3-21 当前页面预览效果

由图可见，此时整个页面背景变成了银色。由于还没添加组件元素，所以尚看不出来 flex 布局模型效果。

然后继续添加两个面板组件，WXML(pages/index/index.wxml)代码片段修改如下：

```
1.  <view class = 'container'>
2.    <view class = 'topPanel'>
3.    </view>
4.    <view class = 'servicePanel'>
5.      <view class = 'serviceTitle'>腾讯服务</view>
```

```
6.        <view class = 'serviceBlocks'></view>
7.      </view>
8.    </view>
```

WXSS(pages/index/index.wxss)代码片段如下：

```
1.  /*2 面板 1: 顶端状态栏 */
2.  .topPanel {
3.    height: 300rpx;                       /* 高度 */
4.    background-color: #686F79;            /* 背景颜色为灰色 */
5.    display: flex;                        /* flex 布局模型 */
6.    flex-direction: row;                  /* 水平布局 */
7.  }
8.  /*3 面板 2: "腾讯服务"栏 */
9.  .servicePanel {
10.   min-height: 600rpx;                   /* 最小高度 */
11.   background-color: white;              /* 背景颜色为白色 */
12.   margin: 20rpx 0;                      /* 上下外边距 20rpx,左右 0 */
13. }
14. /*3-1 面板 2: 第一行标题样式 */
15. .serviceTitle {
16.   padding: 20rpx;                       /* 四周内边距 20rpx */
17.   border: 1rpx solid silver;            /* 1rpx 宽的银色实线边框 */
18.   font-size: 30rpx;                     /* 字号为 30rpx 大小 */
19.   color: gray;                          /* 字体颜色为灰色 */
20. }
21. /*3-2 面板 2: 九宫格区域样式 */
22. .serviceBlocks {
23.   display: flex;                        /* flex 布局模型 */
24.   flex-direction: row;                  /* 水平布局 */
25.   flex-wrap: wrap;                      /* 允许换行 */
26. }
```

当前效果如图 3-22 所示。

由图可见，此时可以显示两个面板的布局位置、背景颜色以及面板 2 的标题。

2）面板 1 方格设计

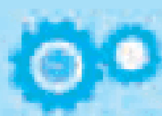

视频讲解

接下来制作面板 1 的方格内容，以其中左边第一个方格“收付款”内容为例，WXML(pages/index/index.wxml)代码片段修改如下：

```
1.  <view class = 'container'>
2.    <view class = 'topPanel'>
3.      <view class = 'box1'>
4.        <image src = '/images/top/money.png'></image>
5.        <text>收付款</text>
6.      </view>
7.    </view>
8.    …
9.  </view>
```

WXSS(pages/index/index.wxss)代码片段如下：

```
1.  /*2-1 面板 1: 方格样式 */
2.  .box1 {
3.    display: flex;                        /* flex 布局模型 */
4.    flex-direction: column;               /* 垂直布局 */
```



```
5.     align-items: center;             /* 水平方向居中 */
6.     width: 33%;                      /* 宽度约占屏幕的 1/3 */
7.     height: 250rpx;                  /* 高度 */
8.   }
9.   /* 2-2 面板 1: 方格内图标样式 */
10.  .box1 image{
11.    width: 110rpx;                   /* 宽度 */
12.    height: 110rpx;                  /* 高度 */
13.    margin: 20rpx;                   /* 四周外边距均为 20rpx */
14.  }
15.  /* 2-3 面板 1: 方格内文本样式 */
16.  .box1 text{
17.    text-align: center;              /* 文本居中 */
18.    color: white;                    /* 字体颜色为白色 */
19.    font-size: 35rpx;                /* 字号为 35rpx 大小 */
20.  }
```

当前效果如图 3-23 所示。



图 3-22　当前页面预览效果



图 3-23　面板 1 预览效果

由图可见，此时可以显示面板 1 的“收付款”方格内容，包括图标和文本。用同样的方式追加其他方格即可实现完整效果。当然也可以暂时不追加，使用 3.2.4 节介绍的方法减少工作量。

3）面板 2 方格设计



视频讲解

接下来制作面板 2 的方格内容，以其中左边第一个方格“信用卡还款”内容为例，WXML(pages/index/index.wxml)代码片段修改如下：

```
1.  <view class = 'container'>
2.      …
3.    <view class = 'servicePanel'>
```

```
4.       <view class = 'serviceTitle'>腾讯服务</view>
5.       <view class = 'serviceBlocks'>
6.         <view class = 'box2'>
7.           <image src = '/images/service/cardRepay.png'></image>
8.           <text>信用卡还款</text>
9.         </view>
10.      </view>
11.    </view>
12.  </view>
```

WXSS(pages/index/index.wxss)代码片段如下：

```
1.  /*3-2-1 面板2：九宫格区域方格样式*/
2.  .box2 {
3.    border: 1rpx solid silver;            /*1rpx宽的银色实线边框*/
4.    display: flex;                        /*flex布局模型*/
5.    flex-direction: column;               /*垂直布局*/
6.    align-items: center;                  /*水平方向居中*/
7.    justify-content: center;              /*垂直方向居中*/
8.    width: 33%;                           /*宽度约占屏幕的1/3*/
9.    height: 230rpx;                       /*高度230rpx*/
10. }
11. /*3-2-2 面板2：方格内图标*/
12. .box2 image {
13.   width: 90rpx;                         /*宽度*/
14.   height: 90rpx;                        /*高度*/
15. }
16. /*3-2-3 面板2：方格内文本*/
17. .box2 text {
18.   font-size: 30rpx;   /*字号为30rpx大小*/
19. }
```

当前效果如图3-24所示。

由图可见，此时可以显示面板2的“信用卡还款”方格内容。用同样的方式追加其他方格即可实现完整效果。当然也可以暂时不追加，使用3.2.4节介绍的方法减少工作量。



图3-24 面板2预览效果

3.2.4 逻辑实现



视频讲解

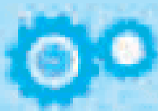
1 面板1的逻辑实现

1）使用动态数据展示方格

由于所有方格的内容布局都是统一的，可以考虑使用wx:for属性配合动态数组渲染全部列表项，以减少WXML页面的代码量。

修改WXML(pages/index/index.wxml)页面代码如下：

```
1.  <view class = 'container'>
2.    <view class = 'topPanel'>
3.      <view class = 'box1' wx:for = '{{array1}}' wx:key = 'array1_{{index}}'>
4.        <image src = '{{item.icon}}'></image>
```

```
5.          <text>{{item.text}}</text>
6.        </view>
7.      </view>
8.      …
9.    </view>
```

上述代码表示使用 wx:for 属性循环显示全部方格，其中{{array1}}数组用于表示 3 个方格，方格区域中的图标和文本分别命名为 icon、text。这里均为自定义名称，开发者可以自行更改。



视频讲解

2）补充数组完整信息

在 index.js 的 data 属性中添加 array1 数组，JS 文件(pages/index/index.js)代码如下：

```
1.  Page({
2.    data: {
3.      //面板 1 的九宫格数组
4.      array1: [
5.        { icon: '/images/top/money.png', text: '收付款' },
6.        { icon: '/images/top/balance.png', text: '零钱\n0.00' },
7.        { icon: '/images/top/cards.png', text: '银行卡' },
8.      ]
9.    }
10. })
```

当前效果如图 3-25 所示。



视频讲解

2 面板 2 的逻辑实现

1）使用动态数据展示方格

接下来继续使用 wx:for 属性配合动态数组渲染全部列表项，以减少 WXML 页面的代码量。

修改 WXML(pages/index/index.wxml)页面代码如下：

```
1.  <view class='container'>
2.    …
3.    <view class='servicePanel'>
4.      <view class='serviceTitle'>腾讯服务</view>
5.      <view class='serviceBlocks'>
6.        <view class='box2' wx:for='{{array2}}' wx:key='array2_{{index}}'>
7.          <image src='{{item.icon}}'></image>
8.          <text>{{item.text}}</text>
9.        </view>
10.     </view>
11.   </view>
12. </view>
```

上述代码表示使用 wx:for 属性循环显示全部方格，其中{{array2}}数组用于表示 8 个方格，方格区域中的图标和文本分别命名为 icon、text。这里均为自定义名称，开发者可以自行更改。



视频讲解

2）补充数组完整信息

在 index.js 的 data 属性中添加 array2 数组，JS 文件(pages/index/index.js)代码如下：

```
1.  Page({
2.    data: {
3.      //面板 1 的九宫格数组
4.     …,
5.      //面板 2 的九宫格数组
6.      array2: [
7.        { icon: '/images/service/cardRepay.png', text: '信用卡还款' },
8.        { icon: '/images/service/mobileTopup.png', text: '手机充值' },
9.        { icon: '/images/service/wealth.png', text: '理财通' },
10.       { icon: '/images/service/utilities.png', text: '生活缴费' },
11.       { icon: '/images/service/qqCoins.png', text: 'Q 币充值' },
12.       { icon: '/images/service/publicService.png', text: '城市服务' },
13.       { icon: '/images/service/charity.png', text: '腾讯公益' },
14.       { icon: '/images/service/insurance.png', text: '保险服务' }
15.     ]
16.   }
17. })
```

此时就已全部完成，保存后重新预览项目，最终效果图如图 3-26 所示。



图 3-25　面板 1 预览效果



图 3-26　最终效果图

3.2.5　完整代码展示

app.json 文件的完整代码如下：

```
1.  {
2.    "pages": [
3.      "pages/index/index"
4.    ]
5.  }
```

JSON 文件(pages/index/index.json)的完整代码如下：

```
1.  {
2.    "navigationBarTitleText": "钱包",
3.    "navigationBarBackgroundColor":"#686F79"
4.  }
```

WXML 文件(pages/index/index.wxml)的完整代码如下：

```
1.  <view class = 'container'>
2.    <view class = 'topPanel'>
3.      <view class = 'box1' wx:for = '{{array1}}' wx:key = 'array1_{{index}}'>
4.        <image src = '{{item.icon}}'></image>
5.        <text>{{item.text}}</text>
6.      </view>
7.    </view>
8.    <view class = 'servicePanel'>
9.      <view class = 'serviceTitle'>腾讯服务</view>
10.     <view class = 'serviceBlocks'>
11.       <view class = 'box2' wx:for = '{{array2}}' wx:key = 'array2_{{index}}'>
12.         <image src = '{{item.icon}}'></image>
13.         <text>{{item.text}}</text>
14.       </view>
15.     </view>
16.   </view>
17. </view>
```

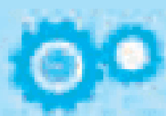
WXSS 文件(pages/index/index.wxss)的完整代码如下：

```
1.  /*1 背景容器样式*/
2.  .container {
3.    height: 100vh;                      /*高度为 100 视窗,写成 100% 无效*/
4.    background-color: silver;           /*背景颜色为银色*/
5.    display: flex;                      /*flex 布局模型*/
6.    flex-direction: column;             /*垂直布局*/
7.  }
8.
9.  /*2 面板 1: 顶端状态栏*/
10. .topPanel {
11.   height: 300rpx;                     /*高度*/
12.   background-color: #686F79;          /*背景颜色为灰色*/
13.   display: flex;                      /*flex 布局模型*/
14.   flex-direction: row;                /*水平布局*/
15. }
16. /*2-1 面板 1: 方格样式*/
17. .box1 {
18.   display: flex;                      /*flex 布局模型*/
19.   flex-direction: column;             /*垂直布局*/
20.   align-items: center;                /*水平方向居中*/
21.   width: 33%;                         /*宽度约占屏幕的 1/3*/
22.   height: 250rpx;                     /*高度*/
23. }
24. /*2-2 面板 1: 方格内图标样式*/
25. .box1 image{
26.   width: 110rpx;                      /*宽度*/
27.   height: 110rpx;                     /*高度*/
```

```
28.   margin: 20rpx;                  /* 四周外边距均为 20rpx */
29. }
30. /* 2-3 面板 1: 方格内文本样式 */
31. .box1 text{
32.   text-align: center;             /* 文本居中 */
33.   color: white;                   /* 字体颜色为白色 */
34.   font-size: 35rpx;               /* 字号为 35rpx 大小 */
35. }
36.
37. /* 3 面板 2: "腾讯服务"栏 */
38. .servicePanel {
39.   min-height: 600rpx;             /* 最小高度 */
40.   background-color: white;        /* 背景颜色为白色 */
41.   margin: 20rpx 0;                /* 上下外边距 20rpx,左右 0 */
42. }
43. /* 3-1 面板 2: 第一行标题样式 */
44. .serviceTitle {
45.   padding: 20rpx;                 /* 四周内边距 20rpx */
46.   border: 1rpx solid silver;      /* 1rpx 宽的银色实线边框 */
47.   font-size: 30rpx;               /* 字号为 30rpx 大小 */
48.   color: gray;                    /* 字体颜色为灰色 */
49. }
50. /* 3-2 面板 2: 九宫格区域样式 */
51. .serviceBlocks {
52.   display: flex;                  /* flex 布局模型 */
53.   flex-direction: row;            /* 水平布局 */
54.   flex-wrap: wrap;                /* 允许换行 */
55. }
56. /* 3-2-1 面板 2: 九宫格区域方格样式 */
57. .box2 {
58.   border: 1rpx solid silver;      /* 1rpx 宽的银色实线边框 */
59.   display: flex;                  /* flex 布局模型 */
60.   flex-direction: column;         /* 垂直布局 */
61.   align-items: center;            /* 水平方向居中 */
62.   justify-content: center;        /* 垂直方向居中 */
63.   width: 33%;                     /* 宽度约占屏幕的 1/3 */
64.   height: 230rpx;                 /* 高度 230rpx */
65. }
66. /* 3-2-2 面板 2: 方格内图标 */
67. .box2 image {
68.   width: 90rpx;                   /* 宽度 */
69.   height: 90rpx;                  /* 高度 */
70. }
71. /* 3-2-3 面板 2: 方格内文本 */
72. .box2 text {
73.   font-size: 30rpx;               /* 字号的 30rpx 大小 */
74. }
```

JS 文件(pages/index/index.js)的完整代码如下:

```
1. Page({
2.   data: {
3.     //面板 1 的九宫格数组
4.     array1: [
5.       { icon: '/images/top/money.png', text: '收付款' },
6.       { icon: '/images/top/balance.png', text: '零钱\n0.00' },
```



```
7.          { icon: '/images/top/cards.png', text: '银行卡' },
8.        ],
9.        //面板 2 的九宫格数组
10.       array2: [
11.         { icon: '/images/service/cardRepay.png', text: '信用卡还款' },
12.         { icon: '/images/service/mobileTopup.png', text: '手机充值' },
13.         { icon: '/images/service/wealth.png', text: '理财通' },
14.         { icon: '/images/service/utilities.png', text: '生活缴费' },
15.         { icon: '/images/service/qqCoins.png', text: 'Q币充值' },
16.         { icon: '/images/service/publicService.png', text: '城市服务' },
17.         { icon: '/images/service/charity.png', text: '腾讯公益' },
18.         { icon: '/images/service/insurance.png', text: '保险服务' }
19.       ]
20.     }
21. })
```

第4章  Chapter 4

小程序组件·猜数字游戏

本章主要介绍使用小程序组件相关知识制作一款简易的猜数字游戏，系统将随机生成0～100的整数让玩家猜，一共8个回合。

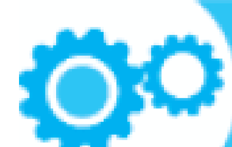
本章学习目标

- 学习使用基础容器< view >；
- 学习使用< form >、< input >和< button >等组件。

效果如图4-1所示。



图4-1　猜数字小游戏效果图

 4.1　项目创建 



视频讲解

本项目创建选择空白文件夹numberGuess，效果如图4-2所示。

单击“新建”按钮完成项目创建，然后准备手动创建页面配置文件。

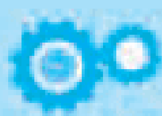
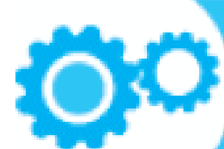
 



图 4-2　小程序项目填写效果示意图



4.2　页面配置





视频讲解

4.2.1　创建页面文件

项目创建完毕后，在根目录中会生成文件夹 pages 用于存放页面文件。

本项目共有 4 个页面文件，介绍如下。

- 首页：index.wxml；
- 开始游戏：game.wxml；
- 游戏规则：rules.wxml；
- 关于我们：about.wxml。

因此需要修改 app.json 文件内 pages 属性中的页面声明，修改后如图 4-3 所示。




```
1  {
2    "pages": [
3      "pages/index/index",
4      "pages/game/game",
5      "pages/about/about",
6      "pages/rules/rules"
7    ]
8  }
```

图 4-3　app.json 页面修改 pages 配置代码

4.2.2　删除和修改文件

具体操作如下：

（1）删除 utils 文件夹及其内部所有内容。

（2）删除 pages 文件夹下的 logs 目录及其内部所有内容。

（3）删除 index.wxml 和 index.wxss 中的全部代码。

（4）删除 index.js 中的全部代码，并且输入关键词 page 找到第二个选项按回车键让其自动补全函数（如图 4-4 所示）。

（5）删除 app.wxss 中的全部代码。

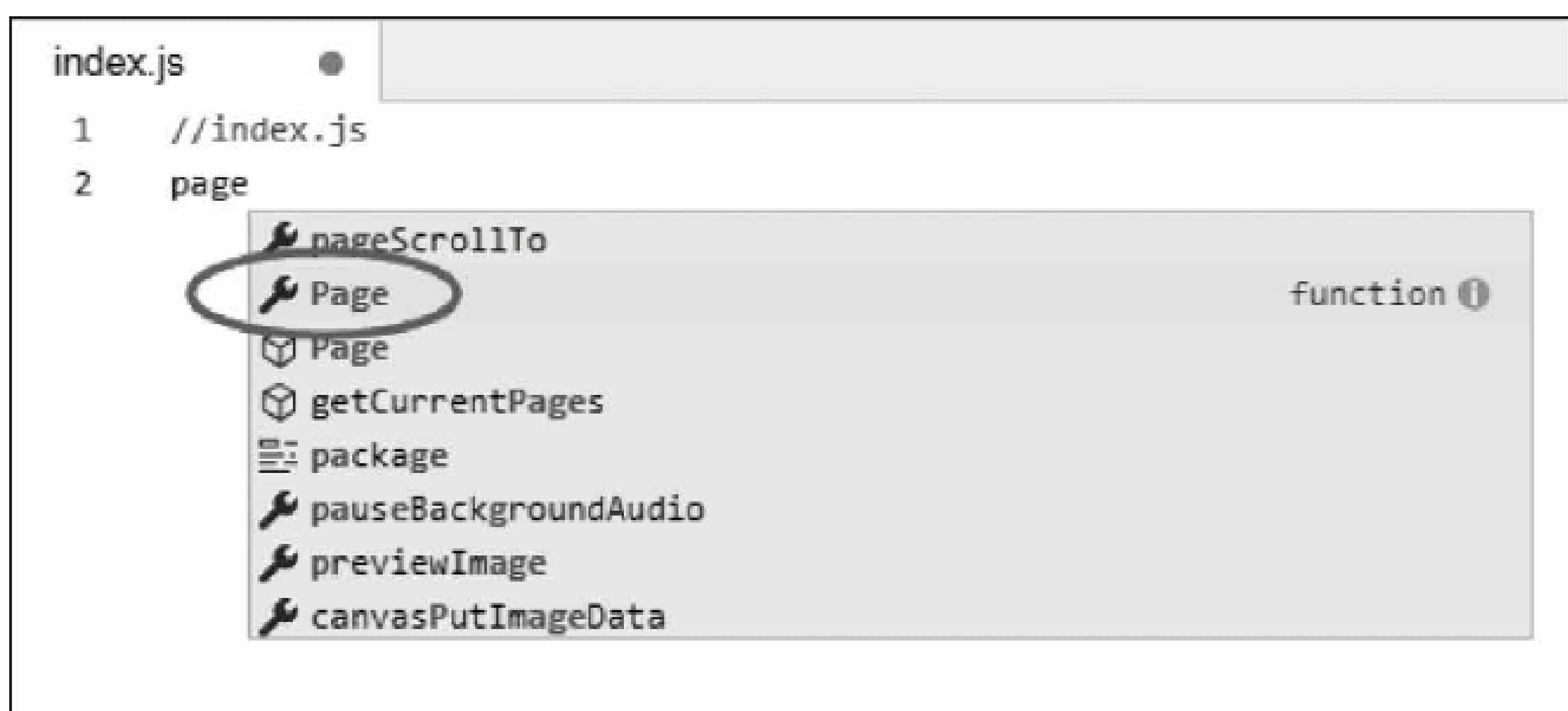


图 4-4　输入关键词创建 Page 函数

（6）删除 app.js 中的全部代码，并且输入关键词 app 找到第一个选项按回车键让其自动补全函数（如图 4-5 所示）。

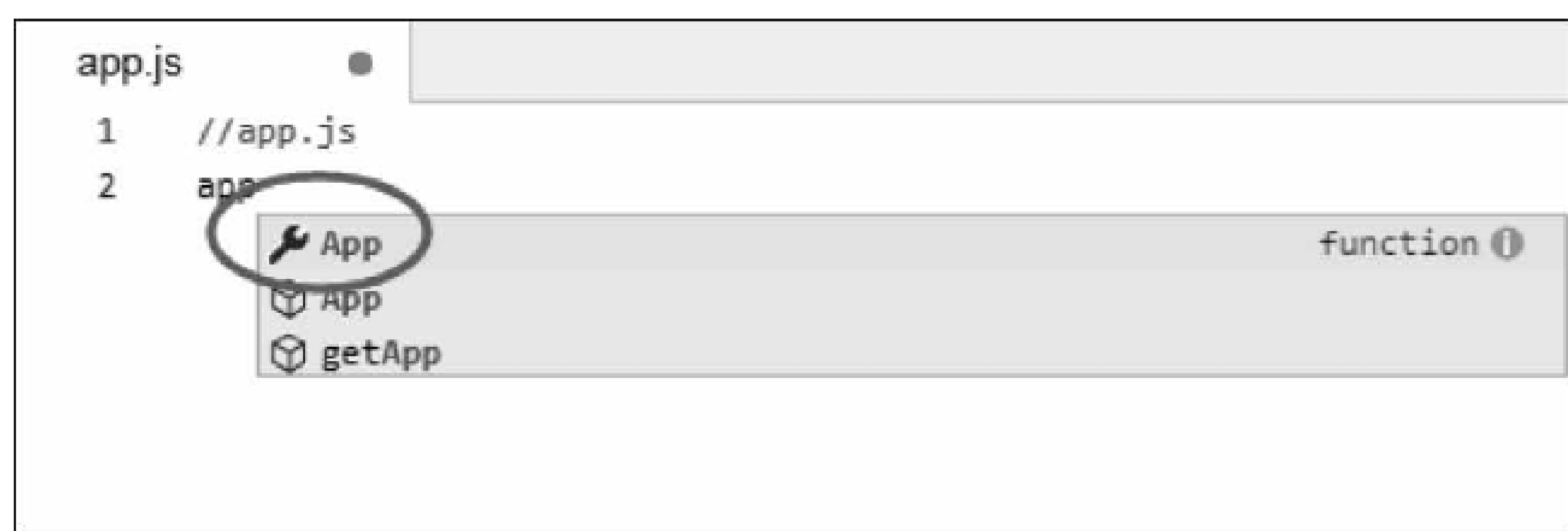
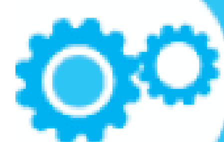


图 4-5　输入关键词创建 App 函数

此时文件配置就全部完成，4.3 节将正式进行页面布局和样式设计。



4.3　视图设计



4.3.1　导航栏设计



视频讲解

小程序默认导航栏就是黑底白字的效果，因此只需要在这 4 个页面的 JSON 文件中分别自定义导航栏标题即可。以 index.json 文件为例，更改后的代码如下：

```
1.  {
2.    "navigationBarTitleText": "猜数字小游戏"
3.  }
```



图 4-6　自定义导航栏效果

上述代码可以更改首页导航栏的标题文本为“猜数字小游戏”，效果如图 4-6 所示。

其他几个页面用同样的方式分别定义标题为“开始游戏”“游戏规则”和“关于我们”，这里不再一一展示代码。

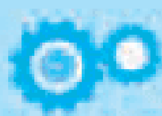
4.3.2　公共样式设计



视频讲解

在 app.wxss 中设置小程序页面的公共样式，代码如下：

```
1.  .container{
2.    display: flex;                    /* flex 模型布局 */
```

```
3.     flex-direction: column;              /* 垂直布局 */
4.     align-items: center;                 /* 水平方向居中 */
5.     height: 100vh;                       /* 高度为 100 视窗,写成 100% 无效 */
6.     justify-content: space-around;       /* 内容调整 */
7.   }
```

上述代码声明了一个名称为 container 的类用于各页面作为底层容器。

4.3.3　页面设计



视频讲解

1 首页设计

首页是菜单选择页,共包含 3 个按钮,具体内容解释如下。

- 开始游戏:点击跳转到"开始游戏"页面;
- 游戏规则:点击跳转到"游戏规则"页面;
- 关于我们:点击跳转到"关于我们"页面。

按钮从上往下排列并且水平方向居中,设计图如图 4-7 所示。

计划使用如下组件。

- 页面整体:<view>组件,并定义 class='container';
- 按钮:<button>组件。

首先定义页面容器(<view>),WXML(pages/index/index.wxml)代码片段如下:

```
1.  <view class='container'>
2.  </view>
```

此时 app.wxss 文件中的样式就会起到作用,由于还没添加组件元素,所以尚看不出来 flex 布局模型效果。

接下来在<view>容器内部依次添加 3 个按钮,WXML(pages/index/index.wxml)代码片段修改如下:

```
1.  <view class='container'>
2.    <button type='primary'>开始游戏</button>
3.    <button type='primary'>游戏规则</button>
4.    <button type='primary'>关于我们</button>
5.  </view>
```

WXSS(pages/index/index.wxss)代码片段如下:

```
1.  /* 按钮样式 */
2.  button{
3.    width: 350rpx;                    /* 宽度 */
4.  }
```

当前效果如图 4-8 所示。

由图可见,此时主菜单页面设计已经完成。

2 "游戏规则"页面设计



视频讲解

"游戏规则"页面只需要包含规则文本<text>组件即可,样式可以由开发者自行设计。

首先定义页面容器(<view>),WXML(pages/rules/rules.wxml)代码片段如下:

```
1.  <view class='container'>
2.  </view>
```

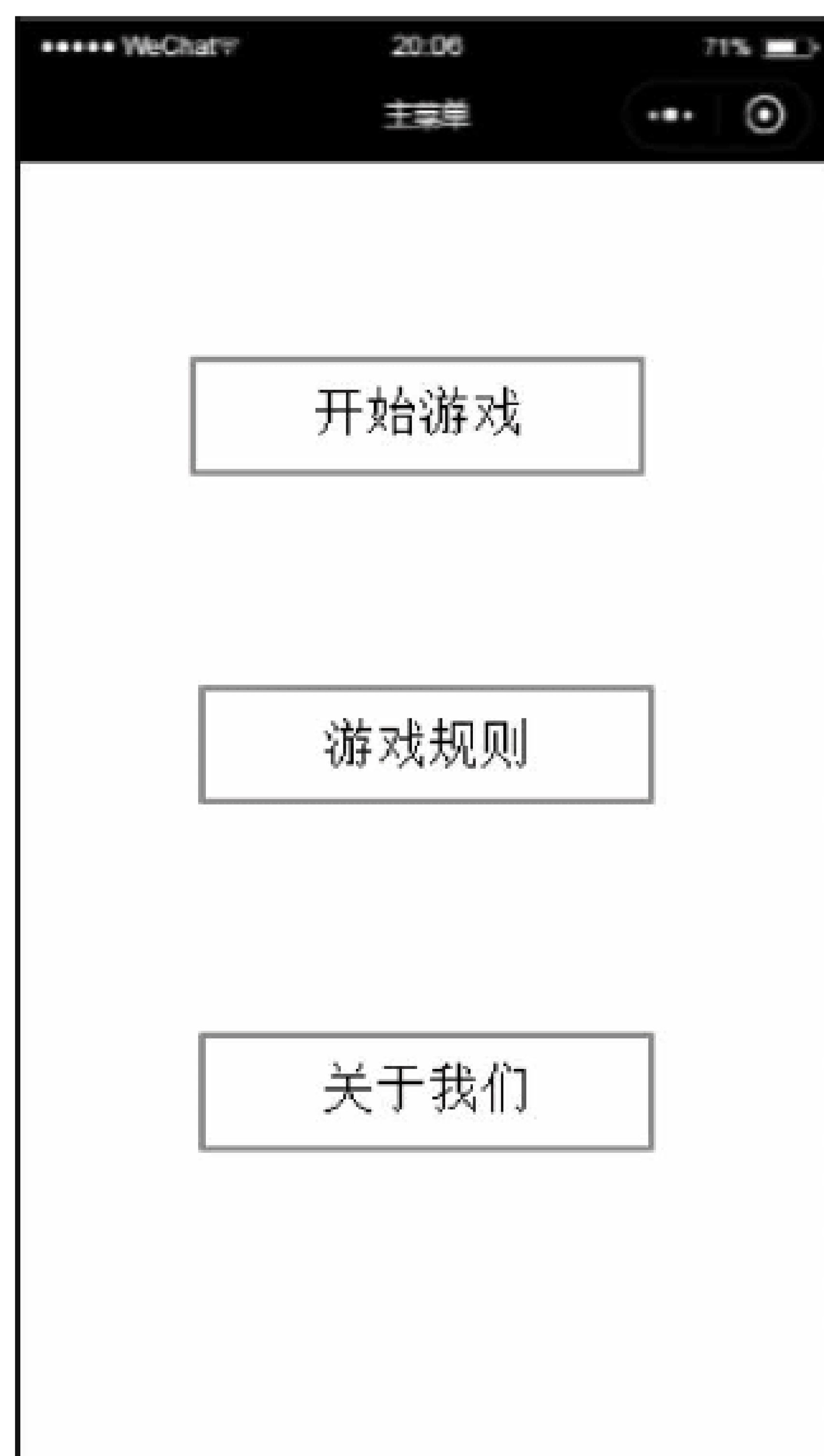


图 4-7　页面设计图



图 4-8　首页预览效果

然后在< view >容器中添加文本，WXML(pages/rules/rules.wxml)代码片段修改如下：

```
1.  <view class='container'>
2.      <text>
3.            1.系统会随机生成一个 0～100 的数字让玩家猜。
4.            2.玩家共有 8 次机会。
5.            3.在 8 次之内猜到则游戏成功。
6.            4.点击"开始游戏"进入游戏画面。
7.      </text>
8.  </view>
```

注意文本组件< text >支持回车换行，还可以使用\n 符号获得一样的显示效果。

当前效果如图 4-9 所示。

由图可见，此时“游戏规则”页面设计已经完成。

3 “关于我们”页面设计



视频讲解

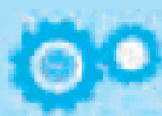
“关于我们”页面同样只需要包含文本组件< text >即可，描述开发者或工作室信息。

首先定义页面容器(< view >)，WXML(pages/about/about.wxml)代码片段如下：

```
1.  <view class='container'>
2.  </view>
```

然后在< view >容器中添加文本，WXML(pages/about/about.wxml)代码片段修改如下：

```
1.  <view class='container'>
2.      <text>×××工作室荣誉出品。</text>
3.  </view>
```



当前效果如图 4-10 所示。

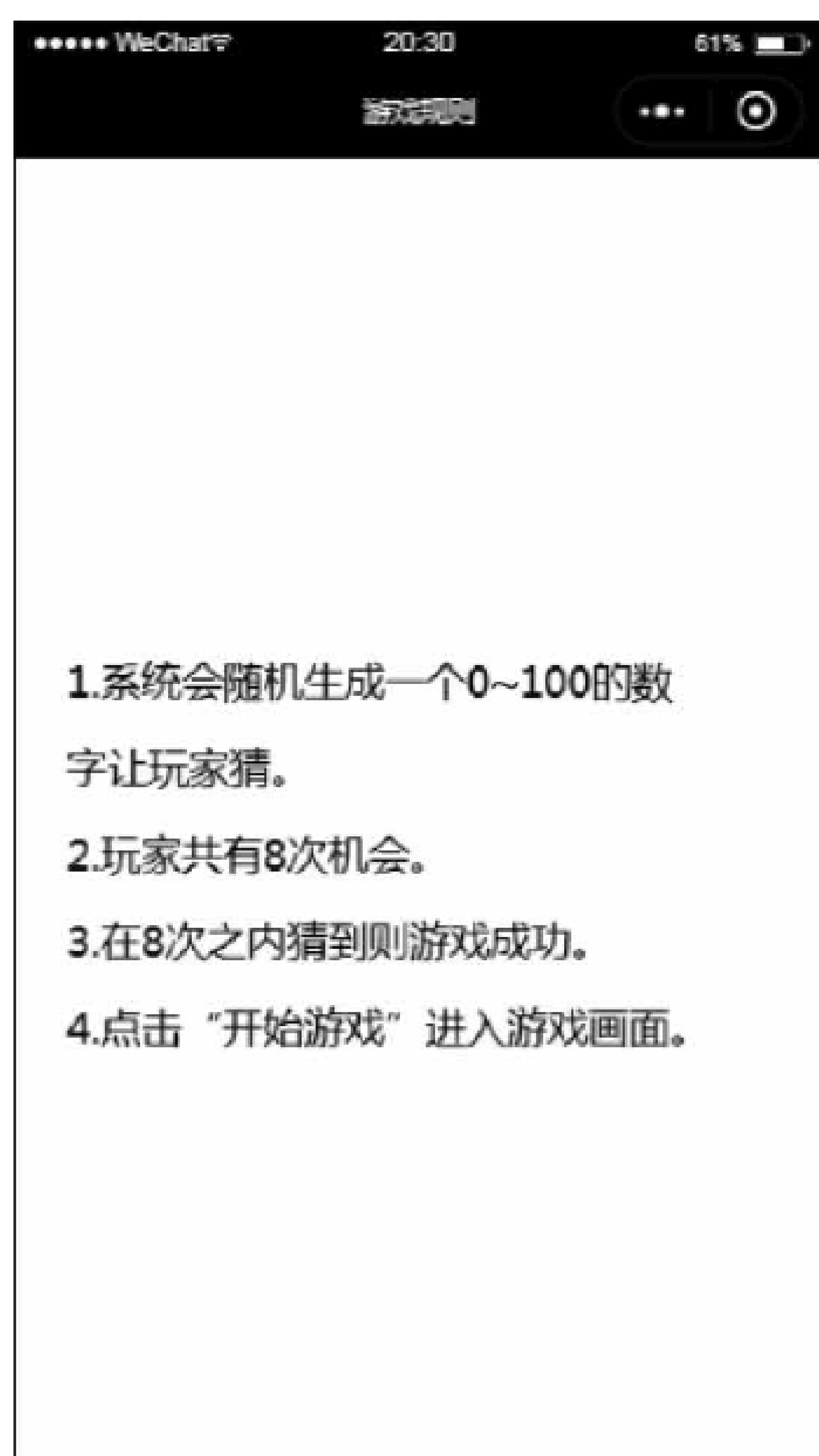


图 4-9 “游戏规则”页面预览效果



图 4-10 “关于我们”页面预览效果

由图可见，此时“关于我们”页面设计已经完成。

4 “开始游戏”页面设计



视频讲解

“开始游戏”页面包含 3 部分内容。

- 顶部欢迎语句：< text >组件；
- 表单：< form >组件；
- 提示语句：< text >组件。

组件从上往下排列并且水平方向居中，设计图如图 4-11 所示。

首先定义页面容器(< view >)，WXML(pages/game/game. wxml)代码片段如下：

```
1.  < view class = 'container'>
2.  </view >
```

然后在< view >容器中添加内容，WXML(pages/game/game. wxml)代码片段修改如下：

```
1.  < view class = 'container'>
2.    < text >欢迎来到猜数字小游戏</text >
3.    < form >
4.        < input type = 'number' placeholder = '请输入 0～100 的数字'></input >
5.        < button type = 'primary' form - type = 'reset'>提交</button >
6.    </form >
7.    < text id = 'tip'></text >
8.  </view >
```

WXSS(pages/game/game. wxss)代码片段如下：

```
1.  /* 文本框 */
2.  input{
3.    border: 1rpx solid green;                /* 1rpx 宽的绿色实线边框 */
```

```
4.     margin: 30rpx 0;                    /* 上下外边距 30rpx */
5.     height: 90rpx;                      /* 高度 */
6.     border-radius: 20rpx;               /* 圆角边框 */
7.   }
8.
9.   /* 提示框 */
10.  #tip{
11.    height:800rpx;                      /* 固定高度 */
12.  }
```

当前效果如图 4-12 所示。

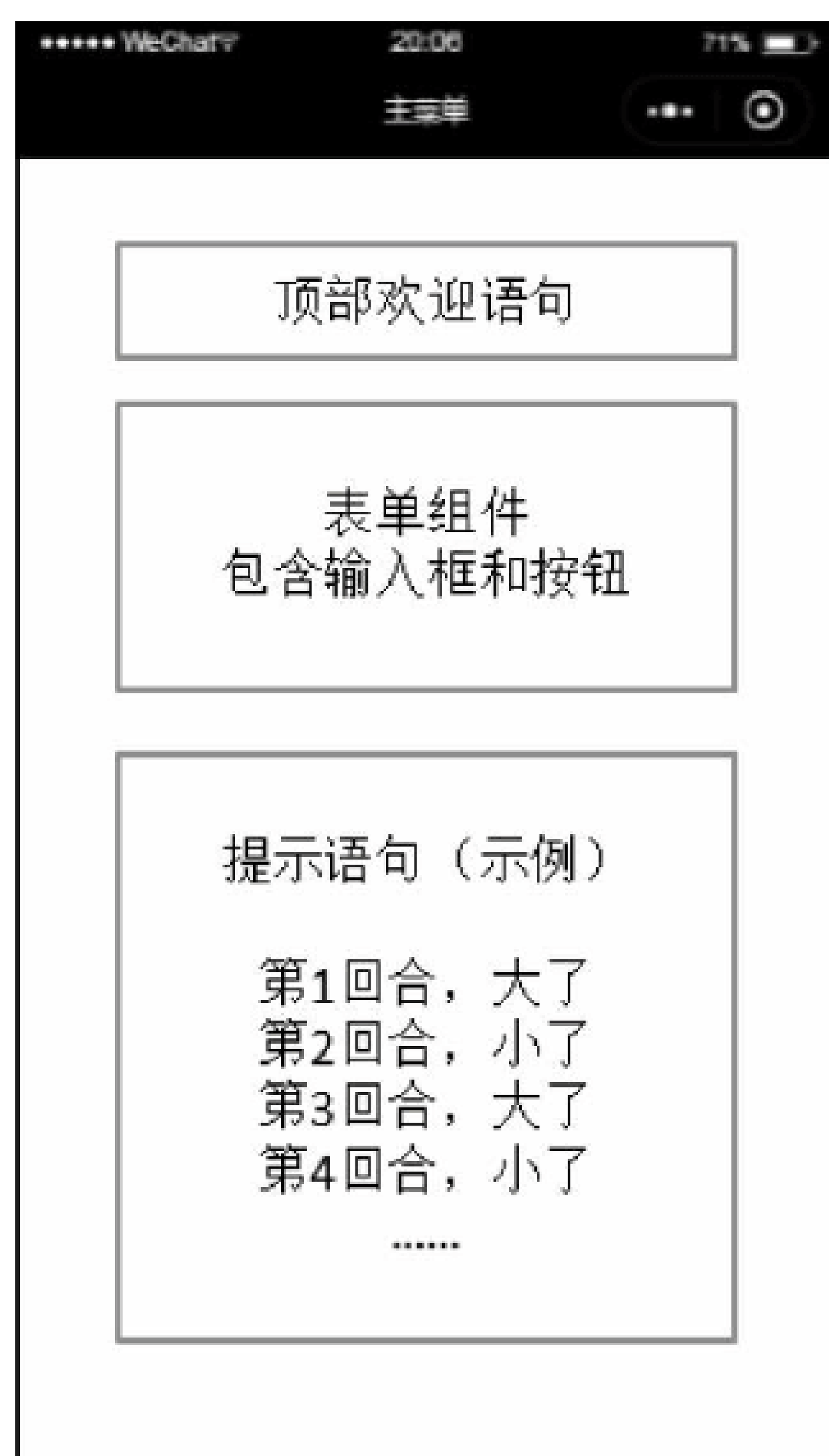
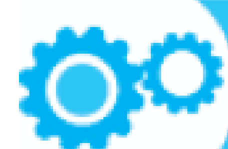


图 4-11 页面设计图



图 4-12 “开始游戏”页面预览效果

由图可见，此时“开始游戏”页面设计已经完成。



4.4 逻辑实现



4.4.1 游戏页面的逻辑实现



视频讲解

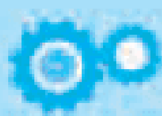
1 游戏初始化

游戏初始化时需要规定以下内容，顺序不限。

- 正确答案 answer：随机生成一个 0～100 的数字；
- 回合数 count：0；
- 提示语句 tip：空字符串''；
- 用户猜的数字 x：－1；
- 游戏状态 isGameStart：true。

首先在 JS 文件(pages/game/game.js)中创建 initial 函数用于初始化数据，代码片段如下：

```
1.  Page({
2.    …
```



```
3.     / **
4.       * 数据初始化
5.       * /
6.     initial: function() {
7.       this.setData({
8.         answer: Math.round(Math.random() * 100),          //随机数
9.         count: 0,                                         //回合数
10.        tip: '',                                          //提示语句
11.        x: -1,                                            //用户猜的数字
12.        isGameStart: true                                 //游戏已经开始
13.      });
14.    },
15.    …
16. })
```

然后在 onLoad 函数中进行调用，JS 文件（pages/game/game.js）代码片段如下：

```
1.     onLoad: function(options) {
2.       this.initial();
3.     },
```

此时可以在页面加载后开始游戏。



视频讲解

2 获取用户输入的数字

为输入框添加 bindinput 事件，修改后 WXML（pages/game/game.wxml）页面代码如下：

```
1.  <view class = 'container'>
2.    <text>欢迎来到猜数字小游戏</text>
3.    <form>
4.        <input bindinput = 'getNumber' type = 'number' placeholder = '请输入 0~100 的数字'></input>
5.        <button type = 'primary' form-type = 'reset'>提交</button>
6.    </form>
7.    <text id = 'tip'></text>
8.  </view>
```

上述代码表示当点击按钮时文本框失去焦点并触发自定义的 getNumber 函数。

在 JS 文件（pages/game/game.js）中添加 getNumber 函数，代码片段如下：

```
1.  Page({
2.     …
3.     / **
4.       * 获取用户输入的数字
5.       * /
6.     getNumber: function(e) {
7.       this.setData({x:e.detail.value})
8.     },
9.     …
10. })
```

3 游戏过程



视频讲解

为按钮添加 bindtap 事件，修改后 WXML（pages/game/game.wxml）页面代码如下：

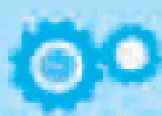
```
1.  <view class = 'container'>
2.    <text>欢迎来到猜数字小游戏</text>
3.    <form>
```

```
4.         <input bindblur='getNumber' type='number' placeholder='请输入 0～100 的数字'></input>
5.         <button type='primary' form-type='reset' bindtap='guess'>提交</button>
6.     </form>
7.     <text id='tip'></text>
8.   </view>
```

上述代码表示当点击按钮时触发自定义的 guess 函数。

在 JS 文件(pages/game/game.js)中添加 guess 函数,代码片段如下:

```
1.   Page({
2.     …
3.   /**
4.      * 本回合开始猜数字
5.      */
6.     guess: function() {
7.       //获取用户本回合填写的数字
8.       let x = this.data.x;
9.       //重置 x 为未获得新数字状态
10.      this.setData({x: -1});
11.
12.      if (x < 0) {
13.        wx.showToast({
14.          title: '不能小于 0',
15.        });
16.      } else if (x > 100) {
17.        wx.showToast({
18.          title: '不能大于 100',
19.        });
20.      } else {
21.        //回合数增加 1
22.        let count = this.data.count + 1;
23.        //获取当前提示信息
24.        let tip = this.data.tip;
25.        //获取正确答案
26.        let answer = this.data.answer;
27.
28.        if (x == answer) {
29.          tip += '\n 第' + count + '回合: ' + x + ',猜对了!';
30.          this.setData({isGameStart: false});              //游戏结束
31.        } else if (x > answer) {
32.          tip += '\n 第' + count + '回合: ' + x + ',大了!';
33.        } else {
34.          tip += '\n 第' + count + '回合: ' + x + ',小了!';
35.        }
36.
37.        if (count == 8) {
38.          tip += '\n 游戏结束';
39.          this.setData({ isGameStart: false });            //游戏结束
40.        }
41.
42.        //更新提示语句和回合数
43.        this.setData({
44.          tip: tip,
45.          count: count
46.        });
47.      }
48.    },  …
49.  })
```



运行效果图如图 4-13 所示。



(a) 游戏开始　　(b) 游戏过程

图 4-13　游戏效果图

4 游戏结束



视频讲解

当游戏结束时隐藏输入框和“提交”按钮，只显示“重新开始”按钮。修改后 WXML(pages/game/game.wxml)页面代码如下：

```
1.  < view class = 'container'>
2.    < text >欢迎来到猜数字小游戏</text >
3.    < form >
4.      < block wx:if = '{{isGameStart}}'>
5.        < input bindblur = 'getNumber' type = 'number' placeholder = '请输入 0～100 的数字'>
</input >
6.        < button type = 'primary' form - type = 'reset' bindtap = 'guess'>提交</button >
7.      </block >
8.      < block wx:else >
9.        < button type = 'primary' bindtap = 'restartGame'>重新开始</button >
10.     </block >
11.   </form >
12.   < text id = 'tip'>{{tip}}</text >
13. </view >
```

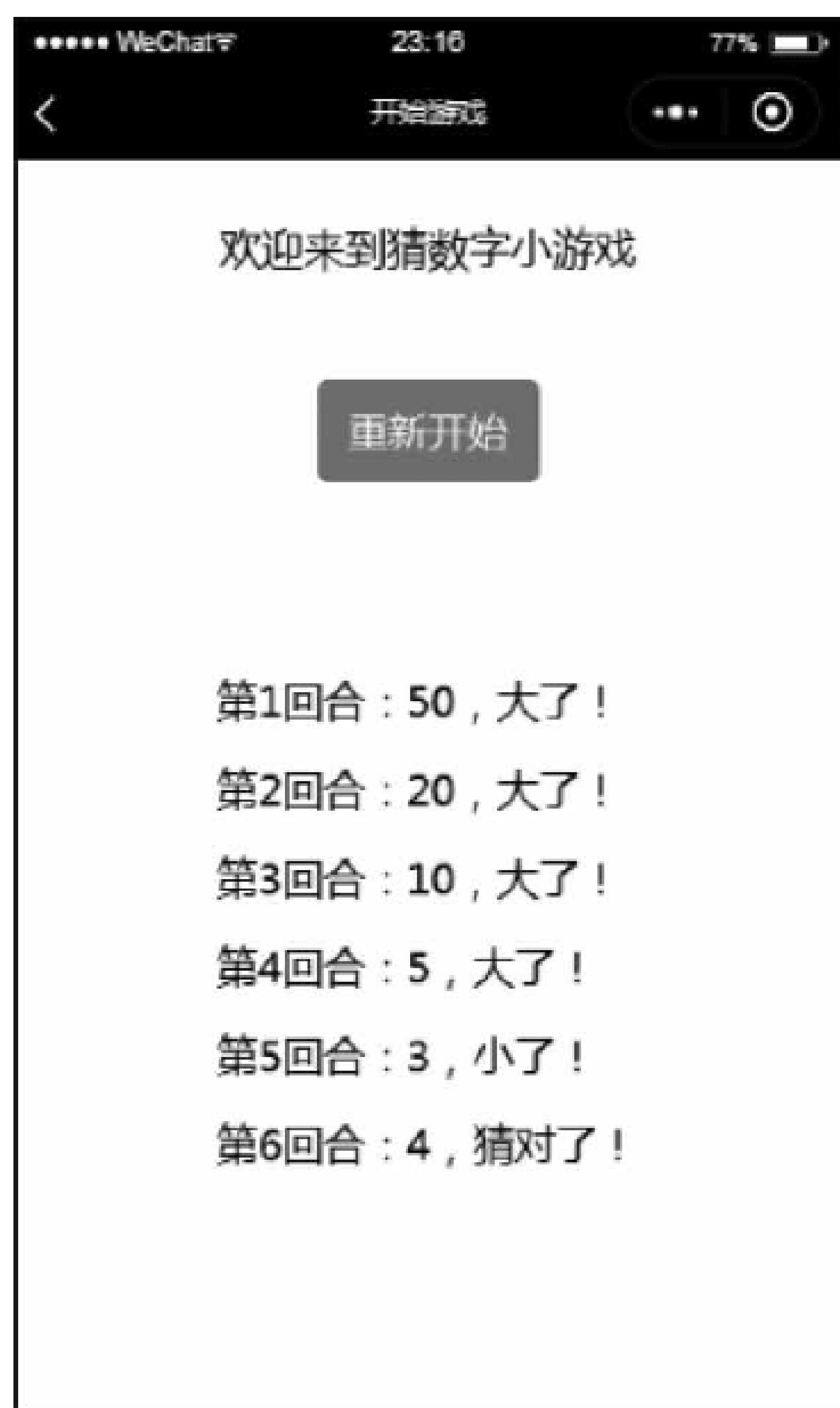
上述代码使用了 wx:if 属性配合< block >代码块形成两种情况，即在游戏中只显示输入框和“提交”按钮，游戏结束时只显示“重新开始”按钮。

在 JS 文件(pages/game/game.js)中添加 restartGame 函数，代码片段如下：

```
1.  Page({
2.    …
3.    / **
4.      *  游戏重新开始
5.      * /
6.    restartGame: function() {
7.      this.initial();
```

```
8.      },
9.      …
10. })
```

运行效果图如图 4-14 所示。



(a) 游戏结束



(b) 游戏重新开始

图 4-14　游戏结束效果图

4.4.2　首页的逻辑实现



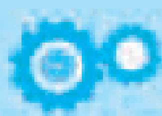
视频讲解

下面将 app.json 中页面的路径位置重新调整，使得 index 为第一个显示的页面，然后为 3 个按钮分别添加 bindtap 事件，WXML 文件(pages/index/index.wxml)代码如下：

```
1.  <view class = 'container'>
2.    <button bindtap = 'goToGame' type = 'primary'>开始游戏</button>
3.    <button bindtap = 'goToRules' type = 'primary'>游戏规则</button>
4.    <button bindtap = 'goToAbout' type = 'primary'>关于我们</button>
5.  </view>
```

JS 文件(pages/index/index.js)代码如下：

```
1.  Page({
2.      …
3.    goToGame(){
4.          wx.navigateTo({
5.          url: '../game/game',
6.          })
7.    },
8.    goToAbout() {
9.      wx.navigateTo({
10.     url: '../about/about',
11.     })
```



```
12.    },
13.    goToRules() {
14.      wx.navigateTo({
15.      url: '../rules/rules',
16.      })
17.    },
18.    …
19. })
```

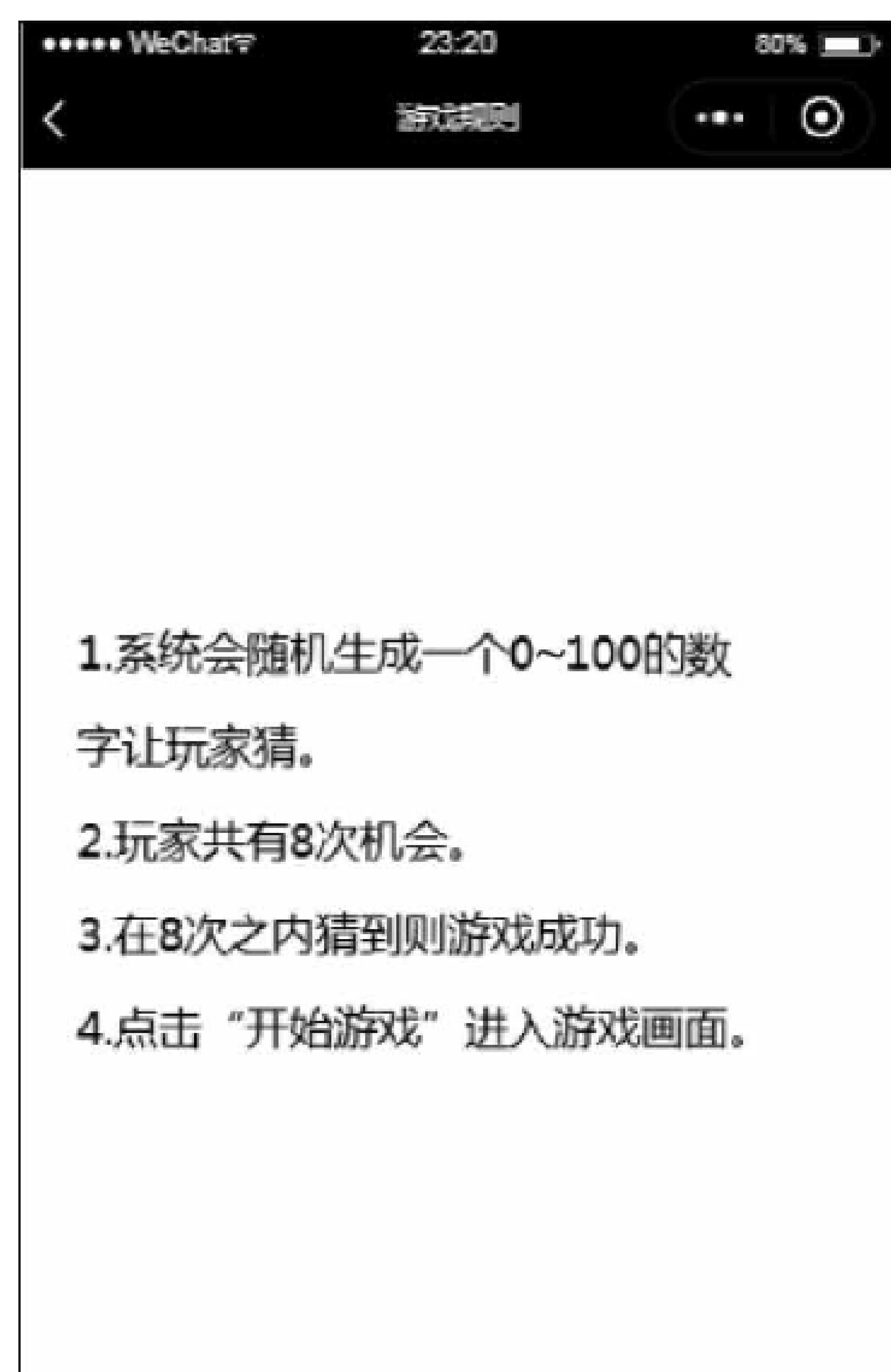
此时已全部完成，保存后重新预览项目，最终效果图如图 4-15 所示。



(a) 首页



(b) 跳转到“开始游戏”页面



(c) 跳转到“游戏规则”页面



(d) 跳转到“关于我们”页面

图 4-15　最终效果图



4.5 完整代码展示



4.5.1 主体文件代码展示

app.json 文件的完整代码如下：

```
1.  {
2.    "pages": ]
3.      "pages/index/index",
4.      "pages/game/game",
5.      "pages/about/about",
6.      "pages/rules/rules"
7.    ]
8.  }
```

app.wxss 文件的完整代码如下：

```
1.  /* 背景容器样式 */
2.  .container{
3.    display: flex;                        /* flex 模型布局 */
4.    flex-direction: column;               /* 垂直布局 */
5.    align-items: center;                  /* 水平方向居中 */
6.    height: 100vh;                        /* 高度为 100 视窗,写成 100% 无效 */
7.    justify-content: space-around;        /* 内容调整 */
8.  }
9.  /* 文本样式 */
10. text{
11.   margin:0 50rpx;                       /* 左右外边距 50rpx */
12.   line-height: 30pt;                    /* 行高 30pt */
13. }
```

4.5.2 首页代码展示

JSON 文件(pages/index/index.json)的完整代码如下：

```
1.  {
2.    "navigationBarTitleText": "猜数字小游戏"
3.  }
```

WXML 文件(pages/index/index.wxml)的完整代码如下：

```
1.  <view class='container'>
2.    <button bindtap='goToGame' type='primary'>开始游戏</button>
3.    <button bindtap='goToRules' type='primary'>游戏规则</button>
4.    <button bindtap='goToAbout' type='primary'>关于我们</button>
5.  </view>
```

WXSS 文件(pages/index/index.wxss)的完整代码如下：

```
1.  /* 按钮样式 */
2.  button{
3.    width: 350rpx;                        /* 宽度 */
4.  }
```

JS 文件(pages/index/index.js)的完整代码如下：

```
1.  Page({
2.    goToGame() {
3.      wx.navigateTo({
4.        url: '../game/game',
5.      })
6.    },
7.    goToAbout() {
8.      wx.navigateTo({
9.        url: '../about/about',
10.     })
11.   },
12.   goToRules() {
13.     wx.navigateTo({
14.       url: '../rules/rules',
15.     })
16.   }
17. })
```

4.5.3 “游戏规则”页面代码展示

JSON 文件(pages/rules/rules.json)的完整代码如下：

```
1.  {
2.    "navigationBarTitleText": "游戏规则"
3.  }
```

WXML 文件(pages/rules/rules.wxml)的完整代码如下：

```
1.  <view class = 'container'>
2.    <text>
3.          1.系统会随机生成一个 0～100 的数字让玩家猜.
4.          2.玩家共有 8 次机会.
5.          3.在 8 次之内猜到则游戏成功.
6.          4.点击"开始游戏"进入游戏画面.
7.    </text>
8.  </view>
```

4.5.4 “关于我们”页面代码展示

JSON 文件(pages/about/about.json)的完整代码如下：

```
1.  {
2.    "navigationBarTitleText": "关于我们"
3.  }
```

WXML 文件(pages/about/about.wxml)的完整代码如下：

```
1.  <view class = 'container'>
2.    <text>×××工作室荣誉出品.</text>
3.  </view>
```

4.5.5 开始游戏(game)代码展示

JSON(pages/game/game.json)完整代码如下:

```
1.  {
2.    "navigationBarTitleText": "开始游戏"
3.  }
```

WXML(pages/game/game.wxml)完整代码如下:

```
1.  <view class = 'container'>
2.    <text>欢迎来到猜数字小游戏</text>
3.
4.    <form>
5.      <block wx:if = '{{isGameStart}}'>
6.        <input bindinput = 'getNumber' type = 'number' placeholder = '请输入 0~100 的数字'></input>
7.        <button type = 'primary' form - type = 'reset' bindtap = 'guess'>提交</button>
8.      </block>
9.      <block wx:else>
10.       <button type = 'primary' bindtap = 'restartGame'>重新开始</button>
11.     </block>
12.   </form>
13.
14.   <text id = 'tip'>{{tip}}</text>
15. </view>
```

WXSS(pages/game/game.wxss)完整代码如下:

```
1.  /* 文本框 */
2.  input{
3.    border: 1rpx solid green;          /* 1rpx 宽的绿色实线边框 */
4.    margin: 30rpx 0;                   /* 上下外边距 30rpx */
5.    height: 90rpx;                     /* 高度 */
6.    border - radius: 20rpx;            /* 圆角边框 */
7.  }
8.
9.  /* 提示框 */
10. #tip{
11.   height:800rpx;                     /* 固定高度 */
12. }
```

JS(pages/game/game.js)完整代码如下:

```
1.  Page({
2.    /**
3.     * 数据初始化
4.     */
5.    initial: function() {
6.      this.setData({
7.        answer: Math.round(Math.random() * 100),   //随机数
8.        count: 0,                                  //回合数
9.        tip: '',                                   //提示语句
10.       x: -1,                                     //用户猜的数字
11.       isGameStart: true                          //游戏已经开始
12.     });
```

```
13.     },
14.     /**
15.      * 获取用户输入的数字
16.      */
17.     getNumber: function(e) {
18.       this.setData({ x : e.detail.value});
19.     },
20.     /**
21.      * 本回合开始猜数字
22.      */
23.     guess: function() {
24.       //获取用户本回合填写的数字
25.       let x = this.data.x;
26.       //重置 x 为未获得新数字状态
27.       this.setData({x: -1});
28. 
29.       if (x < 0) {
30.         wx.showToast({
31.           title: '不能小于 0',
32.         });
33.       } else if (x > 100) {
34.         wx.showToast({
35.           title: '不能大于 100',
36.         });
37.       } else {
38.         //回合数增加 1
39.         let count = this.data.count + 1;
40.         //获取当前提示信息
41.         let tip = this.data.tip;
42.         //获取正确答案
43.         let answer = this.data.answer;
44. 
45.         if (x == answer) {
46.           tip += '\n 第' + count + '回合:' + x + ',猜对了!';
47.           this.setData({isGameStart: false});      //游戏结束
48.         } else if (x > answer) {
49.           tip += '\n 第' + count + '回合:' + x + ',大了!';
50.         } else {
51.           tip += '\n 第' + count + '回合:' + x + ',小了!';
52.         }
53. 
54.         if (count == 8) {
55.           tip += '\n 游戏结束';
56.           this.setData({ isGameStart: false });    //游戏结束
57.         }
58. 
59.         //更新提示语句和回合数
60.         this.setData({
61.           tip: tip,
62.           count: count
63.         });
64.       }
65.     },
66.     /**
67.      * 游戏重新开始
```

```
68.     * /
69.   restartGame: function() {
70.     this.initial();
71.   },
72.   /**
73.     * 生命周期函数 -- 监听页面加载
74.     * /
75.   onLoad: function(options) {
76.     this.initial();
77.   },
78. })
```

应用篇



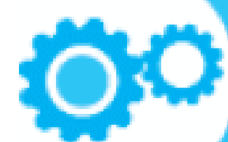
第5章 Chapter 5

小程序网络API·天气查询

本章主要介绍使用小程序网络 API 的相关应用制作一款天气查询小程序。

本章学习目标

- 掌握服务器域名配置和临时服务器部署；
- 掌握 wx. request 接口的用法。



5.1 准备工作





视频讲解

5.1.1 API 密钥申请

本小节主要介绍如何申请获得开源 API 的密钥。这里选择了可以提供全球气象数据服务接口的和风天气 API，其官方网址为“https://www. heweather. com/”(如图 5-1 所示)。



图 5-1 “和风天气”官方主页(访问时间：2018.10.27 10:26)

用户选择“免费用户”类型，使用邮箱进行注册并激活后可以获取三天之内全球各地区的实时天气，免费接口调用流量为 1000 次/天、频率为 200 次/分钟，该数据基本上可以满足读者的开发学习需求。

注册完毕之后可以访问“https://console. heweather. com/my/service”来查看账号信息，用户登录后即可看到开发者申请到的个人认证 key，如图 5-2 所示。

开发者需记录上述页面中的个人认证 key，该信息在小程序发出网络请求时会作为身份识别的标识一并发送给和风天气的第三方服务器。至此，开源 API 的密钥申请就已经顺利完成，读者可以进行 5.1.2 节的学习，了解如何调用 API 获取气象数据。

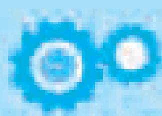




图 5-2　个人认证 key 查询页面(访问时间：2018.10.27 10:27)

5.1.2　API 调用方法



视频讲解

目前免费用户可以调用的最新版接口地址为“https://free-api.heweather.com/s6/”,其服务器节点在中国境内。该接口地址后面追加不同的关键词将获取不同种类的气象数据信息,例如 alarm 为天气自然灾害预警。读者可以访问官方文档(https://dev.heweather.com/docs/api/)了解各类关键词的使用方法。

本示例将选用关键词 weather 进行实况天气数据的获取。实况天气即为当前时间点的天气状况以及温/湿/风/压等气象指数,具体包含体感温度、实测温度、天气状况、风力、风速、风向、相对湿度、大气压强、降水量、能见度等。目前该接口允许查询的城市覆盖范围为全球任意一个城市。

基于关键词 weather 的接口具有两个必填参数和两个可选参数,如表 5-1 所示。

表 5-1　weather 接口参数一览表

参数名称	参数类型	解　释
city	必填参数	用于规定需要查询的城市,可以填入城市名称(国内城市填中文或拼音均可)、城市 ID、IP 地址或经纬度。 例如: city=北京、city=beijing(城市名称) city=CN101010100(城市 ID) city=60.194.130.1(IP 地址) city=120.343,36.088(经纬度)
key	必填参数	需要填入用户的个人认证 key 字符串。接口将通过该数据判断是否为授权用户,并可以进一步判断是否为付费用户。 例如:key=123abc456dfg
lang	可选参数	用于指定数据的语言版本,不添加 lang 参数则默认为简体中文。 例如:lang=en 需要注意的是,国内某些特定数据(例如生活指数、空气质量等)不支持多语言版
unit	可选参数	单位选择,公制(m)或英制(i),默认为公制单位。 例如:unit=i 详见表 5-2“度量衡单位一览表”

其中与 unit 参数相关的公制和英制单位对比如表 5-2 所示。

表 5-2 度量衡单位一览表

数 据 项	公 制 单 位	英 制 单 位
温度	摄氏度：℃	华氏度：℉
风速	公里/小时：km/h	英里/小时：mile/h
能见度	公里：km	英里：mile
大气压强	百帕：hPa	百帕：hPa
降水量	毫米：mm	毫米：mm
PM2.5	微克/立方米：$\mu g/m^3$	微克/立方米：$\mu g/m^3$
PM10	微克/立方米：$\mu g/m^3$	微克/立方米：$\mu g/m^3$
O_3	微克/立方米：$\mu g/m^3$	微克/立方米：$\mu g/m^3$
SO_2	微克/立方米：$\mu g/m^3$	微克/立方米：$\mu g/m^3$
CO	毫克/立方米：mg/m^3	毫克/立方米：mg/m^3
NO_2	微克/立方米：$\mu g/m^3$	微克/立方米：$\mu g/m^3$

注意：部分数据项无论选择何种单位均会使用公制单位。

免费用户调用接口的常见语法格式如下：

```
https://free-api.heweather.com/s6/weather/now?[parameters]
```

其中[parameters]需要替换成使用到的参数，多个参数之间使用 & 符号隔开。

例如，使用拼音查询上海市天气数据的写法如下：

```
https://free-api.heweather.com/s6/weather/now?location=shanghai&key=1234abcd
```

注意，其中 key 的值 1234abcd 为随机填写的内容，请在实际开发中将其替换为真实的个人认证 key，否则接口将无法获取数据。

用户可以直接将这段地址输入到浏览器的地址栏中测试数据返回结果，如图 5-3 所示。

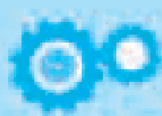
{"HeWeather6":[{"basic":{"cid":"CN101020100","location":"上海","parent_city":"上海","admin_area":"上海","cnty":"中国","lat":"31.23170662","lon":"121.47264099","tz":"+8.00"},"update":{"loc":"2018-10-27 10:45","utc":"2018-10-27 02:45"},"status":"ok","now":{"cloud":"0","cond_code":"100","cond_txt":"晴","fl":"17","hum":"19","pcpn":"0.0","pres":"1024","tmp":"19","vis":"10","wind_deg":"315","wind_dir":"西北风","wind_sc":"1","wind_spd":"4"}}]}

图 5-3 免费天气查询接口返回结果页面(访问时间：2018.10.27 10:59)

由该图可知，指定城市的天气数据返回结果是 JSON 数据格式的文本内容，其中包含的数据是以“名称:值”的形式存放。

为方便用户查看，将图 5-3 返回的数据内容整理格式如下：

```
{
  "HeWeather6":[
    {
      "basic":{
        "cid":"CN101020100",
        "location":"上海",
        "parent_city":"上海",
        "admin_area":"上海",
        "cnty":"中国",
```

```
            "lat":"31.23170662",
            "lon":"121.47264099",
            "tz":" + 8.00"
          },
          "update":{
            "loc":"2018 - 10 - 27 10:45",
            "utc":"2018 - 10 - 27 02:45"
          },
          "status":"ok",
          "now":{
            "cloud":"0",
            "cond_code":"100",
            "cond_txt":"晴",
            "fl":"17",
            "hum":"19",
            "pcpn":"0.0",
            "pres":"1024",
            "tmp":"19",
            "vis":"10",
            "wind_deg":"315",
            "wind_dir":"西北风",
            "wind_sc":"1",
            "wind_spd":"4"
          }
        }
      ]
    }
```

返回的字段说明如表 5-3 所示。

表 5-3 实况天气返回字段说明

basic：基础信息		
参 数	描 述	示 例 值
location	地区/城市名称	海淀
cid	地区/城市 ID	CN101080402
lat	地区/城市纬度	39.956074
lon	地区/城市经度	116.310316
parent_city	该地区/城市的上级城市	北京
admin_area	该地区/城市所属行政区域	北京
cnty	该地区/城市所属国家名称	中国
tz	该地区/城市所在时区	8
update：接口更新时间		
参 数	描 述	示 例 值
loc	当地时间，24 小时制，格式为 yyyy-MM-dd HH:mm	2017/10/25 12:34
utc	UTC 时间，24 小时制，格式为 yyyy-MM-dd HH:mm	2017/10/25 4:34
now：实况天气		
参 数	描 述	示 例 值
fl	体感温度，默认单位为摄氏度	23
tmp	温度，默认单位为摄氏度	21
cond_code	实况天气状况代码	100
cond_txt	实况天气状况描述	晴

续表

参　　数	描　　述	示　例　值
wind_deg	风向 360 角度	305
wind_dir	风向	西北
wind_sc	风力	3
wind_spd	风速，公里/小时	15
hum	相对湿度	40
pcpn	降水量	0
pres	大气压强	1020
vis	能见度，默认单位为公里	10
cloud	云量	23
status：接口状态		
参　　数	描　　述	示　例　值
status	接口状态，具体含义请参考表 5-4“接口状态码及错误码说明”	ok

其中参数 status 的状态码及错误码说明如表 5-4 所示。

表 5-4　接口状态码及错误码说明

代　　码	说　　明
ok	数据正常
invalid key	错误的 key，请检查 key 是否输入以及是否输入有误
unknown location	未知或错误城市/地区
no data for this location	该城市/地区没有所请求的数据
no more requests	超过访问次数，需要等到当月最后一天 24 点(免费用户为当天 24 点)后进行访问次数的重置或升级访问量
param invalid	参数错误，请检查传递的参数是否正确
too fast	超过限定的 QPM，请参考 QPM 说明
dead	无响应或超时，接口服务异常请联系产品开发商
permission denied	无访问权限，没有购买所访问的这部分服务
sign error	签名错误，请参考签名算法

如果接口无法正确地获取数据，可以根据状态码对比该表查询原因。

用户可以根据指定的名称找到对应的数据值，例如在实况天气数据(now)中可以查到当前城市的温度，对应的字段节选如下：

```
"tmp":"19"
```

上述代码表示当前城市的温度为 19℃。

5.1.3　服务器域名配置



视频讲解

每一个小程序在与指定域名地址进行网络通信前都必须将该域名地址添加到管理员后台白名单中，因此本示例需要对域名地址“https://free-api.heweather.com”进行服务器配置。

小程序开发者登录 mp.weixin.qq.com 进入管理员后台，单击“设置”按钮，切换至“开发设置”

面板，在“服务器域名”栏中可以添加或修改需要进行网络通信的服务器域名地址，如图 5-4 所示。

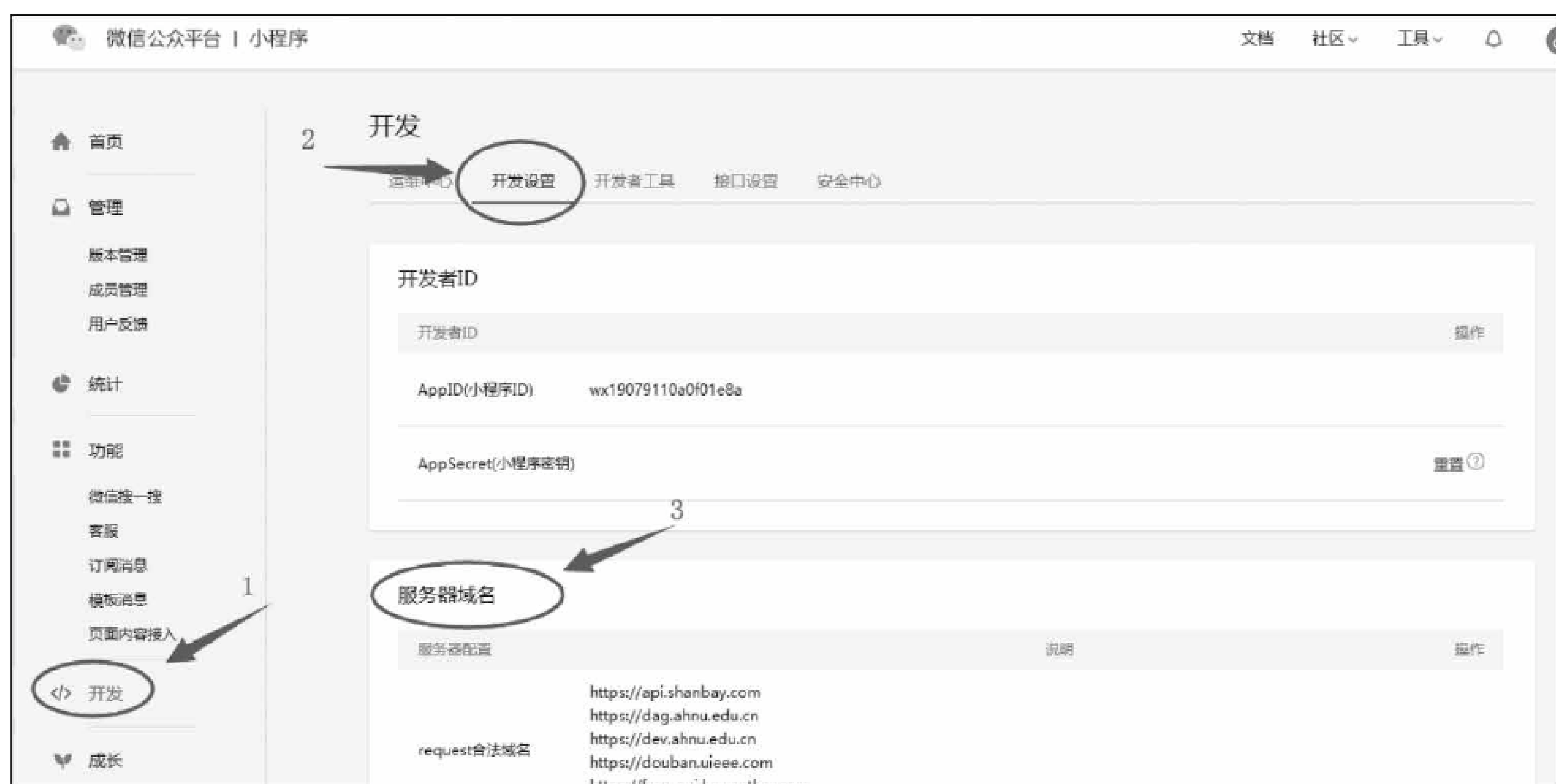


图 5-4　服务器域名配置

将当前需要使用的接口添加到“request 合法域名”中，配置完成后再登录小程序开发工具就允许小程序与指定的服务器域名地址之间的网络通信了，注意每个月只可以申请修改 5 次服务器域名配置。



视频讲解

 5.2　项目创建 

本项目创建选择空白文件夹 weatherDemo，效果如图 5-5 所示。



图 5-5　小程序项目填写效果示意图

单击“新建”按钮完成项目创建，然后准备手动创建页面配置文件。



5.3　页面配置





视频讲解

5.3.1　创建页面文件

项目创建完毕后，在根目录中会生成文件夹 pages 用于存放页面文件。一般来说首页默认命名为 index，表示小程序运行的第一个页面；其他页面名称可以自定义。本项目只需要保留首页(index)即可。

具体操作如下：

(1) 将 app. json 文件内 pages 属性中的“pages/logs/logs”删除，并删除上一行末尾的逗号。

(2) 按快捷键 Ctrl+S 保存当前修改。

5.3.2　删除和修改文件

具体操作如下：

(1) 删除 utils 文件夹及其内部所有内容。

(2) 删除 pages 文件夹下的 logs 目录及其内部所有内容。

(3) 删除 index. wxml 和 index. wxss 中的全部代码。

(4) 删除 index. js 中的全部代码，并且输入关键词 page 找到第二个选项按回车键让其自动补全函数(如图 5-6 所示)。

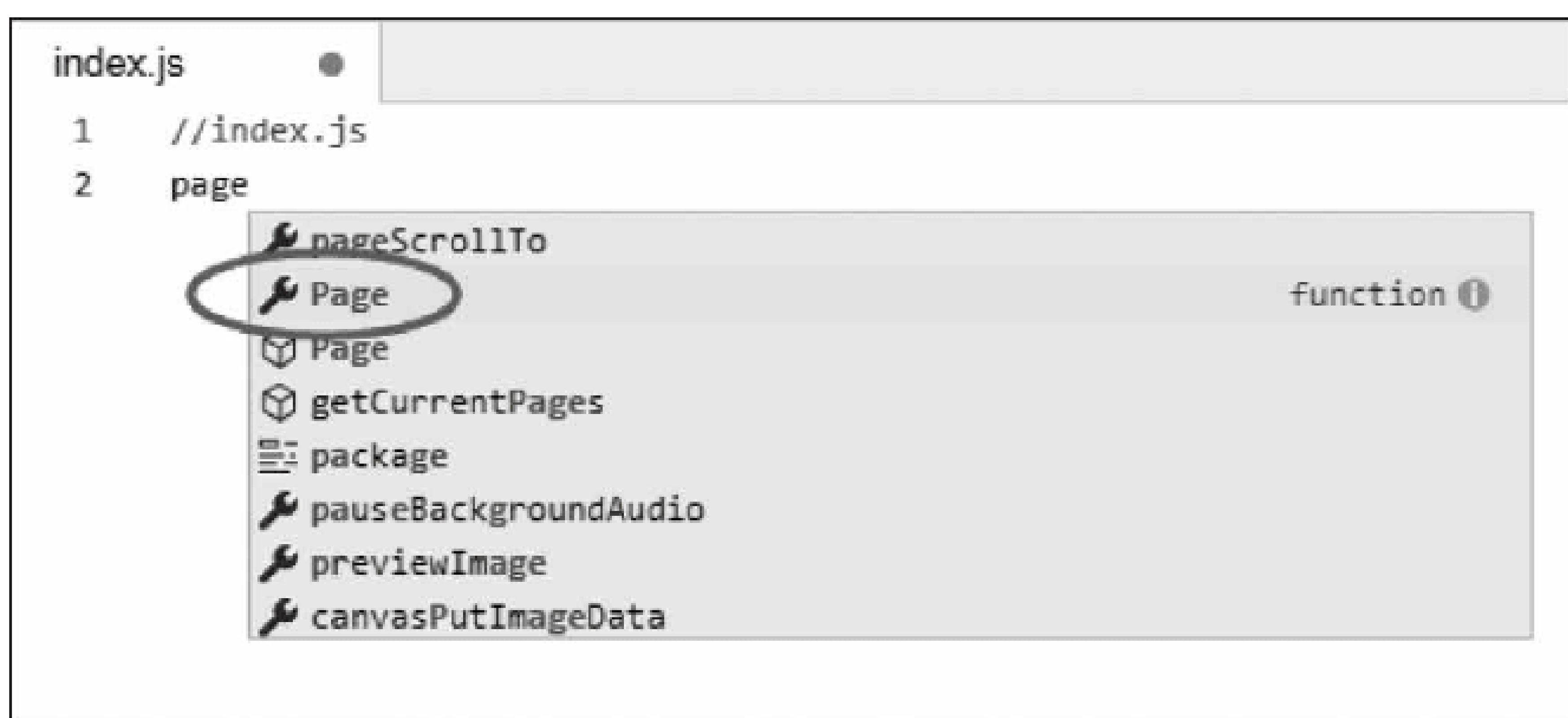


图 5-6　输入关键词创建 Page 函数

(5) 删除 app. wxss 中的全部代码。

(6) 删除 app. js 中的全部代码，并且输入关键词 app 找到第一个选项按回车键让其自动补全函数(如图 5-7 所示)。

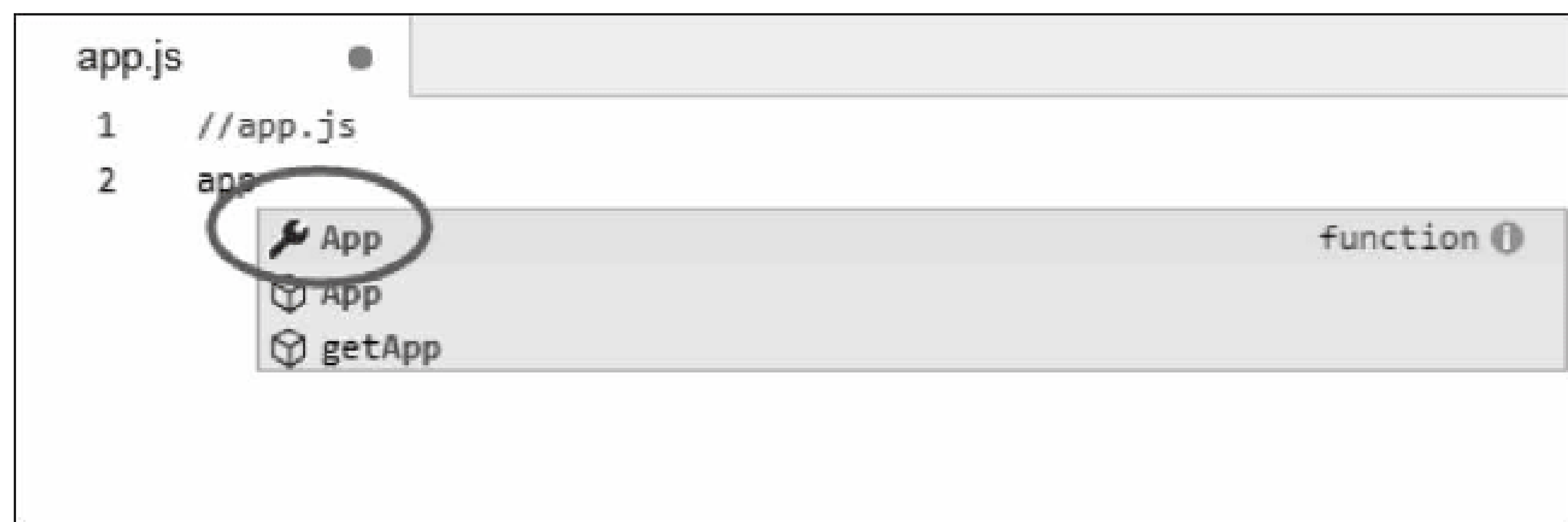
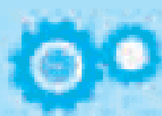


图 5-7　输入关键词创建 App 函数

5.3.3 创建其他文件

接下来创建其他自定义文件，本项目还需要一个文件夹用于存放天气图标素材。文件夹名称由开发者自定义(例如 images)，单击目录结构左上角的＋号创建文件夹并命名为 images。

本项目用到的图标素材共计 75 个，均来源于和风天气官网，图标素材展示如图 5-8 所示。



图 5-8 天气图标素材展示

其中图标文件名为对应的天气代码，扩展名均为.png。需要注意的是，部分图标文件名带有字母 n，表示夜间天气图标，例如 100n.png。

右击目录结构中的 images 文件夹，选择“硬盘打开”，在该文件夹中新建二级目录 weather_icon，然后将图标文件全部复制、粘贴进去。完成后的目录结构如图 5-9 所示。

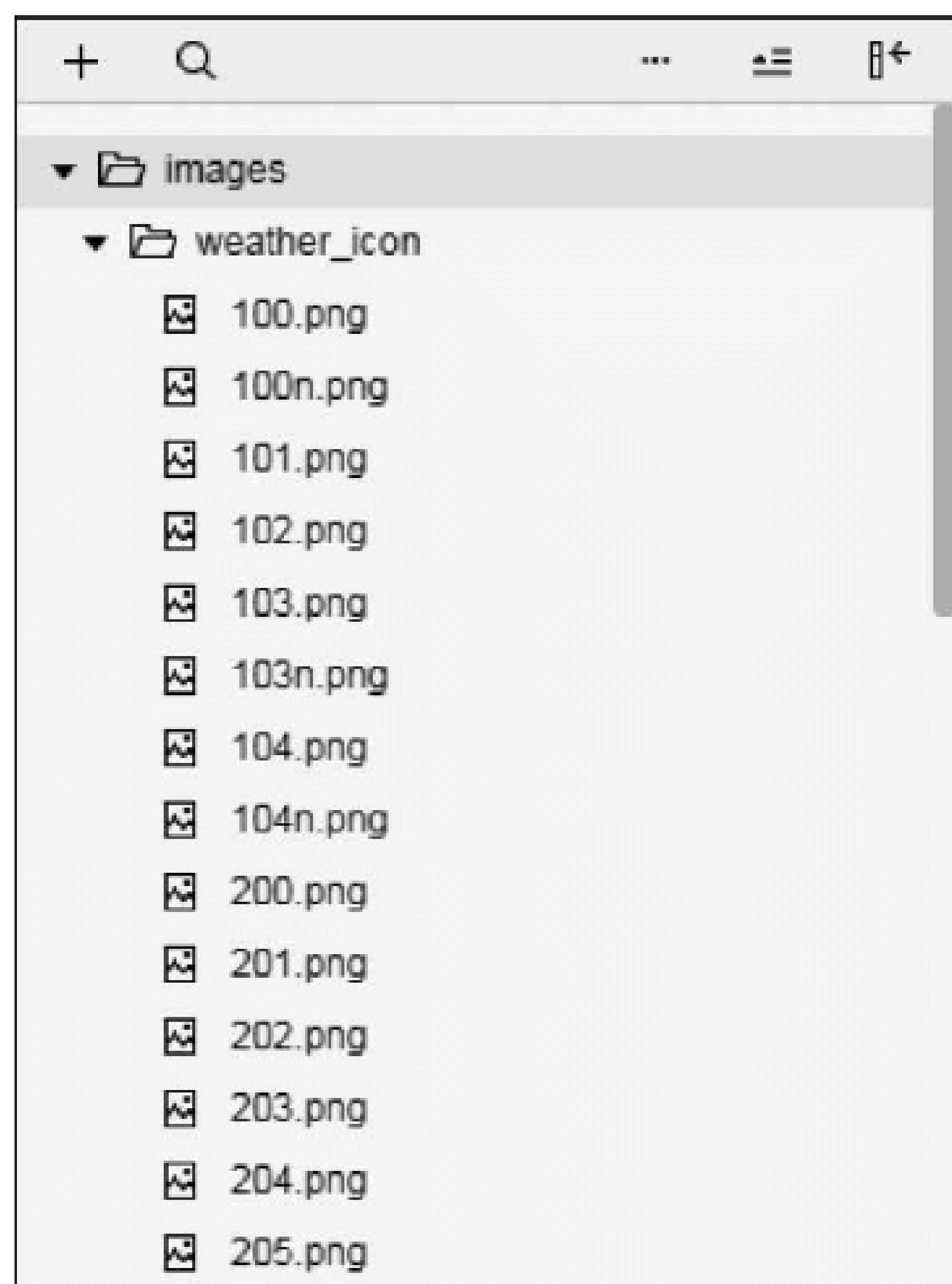


图 5-9 页面文件创建完成

此时文件配置就全部完成,5.4 节将正式进行页面布局和样式设计。



5.4 视图设计





视频讲解

5.4.1 导航栏设计

小程序默认导航栏是黑底白字的效果,因此需要在 app.json 中自定义导航栏标题和背景颜色。更改后的 app.json 文件代码如下:

```
1.  {
2.    "pages": [
3.      "pages/index/index"
4.    ],
5.    "window": {
6.      "navigationBarBackgroundColor": "#3883FA",
7.      "navigationBarTitleText": "今日天气"
8.    }
9.  }
```

上述代码可以更改所有页面的导航栏标题文本为"今日天气"、背景颜色为蓝色(#3883FA)。预览效果如图 5-10 所示。



图 5-10 自定义导航栏效果



视频讲解

5.4.2 页面设计

页面上主要包含 4 个区域,具体内容解释如下。

- 区域 1:地区选择器,用户可以自行选择查询的省、市、区;
- 区域 2:显示当前城市的温度和天气状态的文字说明;
- 区域 3:显示当前城市的天气图标;
- 区域 4:分多行显示其他天气信息,例如湿度、气压、能见度和风向等。

注意,面板之间需要有一定的间隔距离,设计图如图 5-11 所示。

计划使用如下组件。

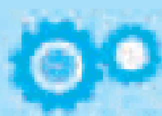
- 页面整体:<view>组件,并定义 class='container';
- 区域 1:<picker>组件;
- 区域 2:<text>组件;
- 区域 3:<image>组件;
- 区域 4:<view>组件,并定义 class='detail';
- 区域 4 内单元行:4 个<view>组件,并定义 class= 'bar';
- 区域 4 内单元格:每行 3 个<view>组件,并定义 class='box'。

1 整体容器设计

首先定义页面容器(<view>),WXML(pages/index/index.wxml)代码片段如下:

```
1.  <view class = 'container'>
2.  </view>
```

在 app.wxss 中设置容器样式,代码片段如下:



```
1.  /* 背景容器样式 */
2.  .container{
3.    height: 100vh;                     /* 高度为 100 视窗,写成 100% 无效 */
4.    display: flex;                     /* flex 布局模型 */
5.    flex-direction: column;            /* 垂直布局 */
6.    align-items: center;               /* 水平方向居中 */
7.    justify-content:space-around;      /* 调整间距 */
8.  }
```

当前效果如图 5-12 所示。

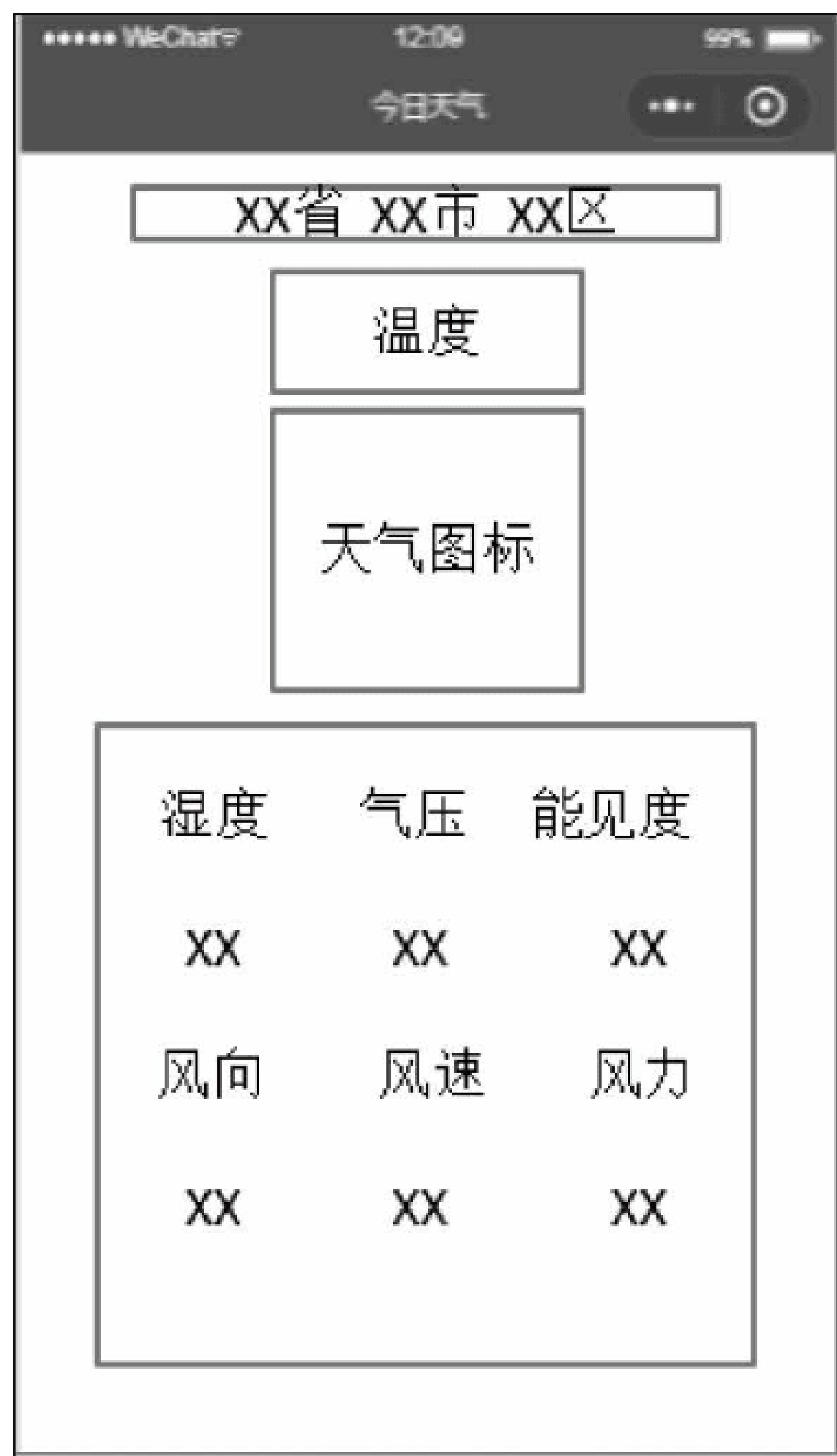


图 5-11　页面设计图



图 5-12　页面预览效果

由于还没有添加组件元素,所以尚看不出来 flex 布局模型效果。

2 区域 1(地区选择器)设计

区域 1 需要使用< picker >组件来实现一个地区选择器,用户点击可切换选择其他城市。

WXML(pages/index/index.wxml)代码片段修改如下:

```
1.  <view class = 'container'>
2.    <!-- 区域 1: 地区选择器 -->
3.    <picker mode = 'region'>
4.      <view>北京市</view>
5.    </picker>
6.  </view>
```

< picker >组件内部是开发者任意填写的一个城市名称,当前效果如图 5-13 所示。

由图可见,点击城市名称时会从底部弹出控件,用户可以进行省、市、区的选择。

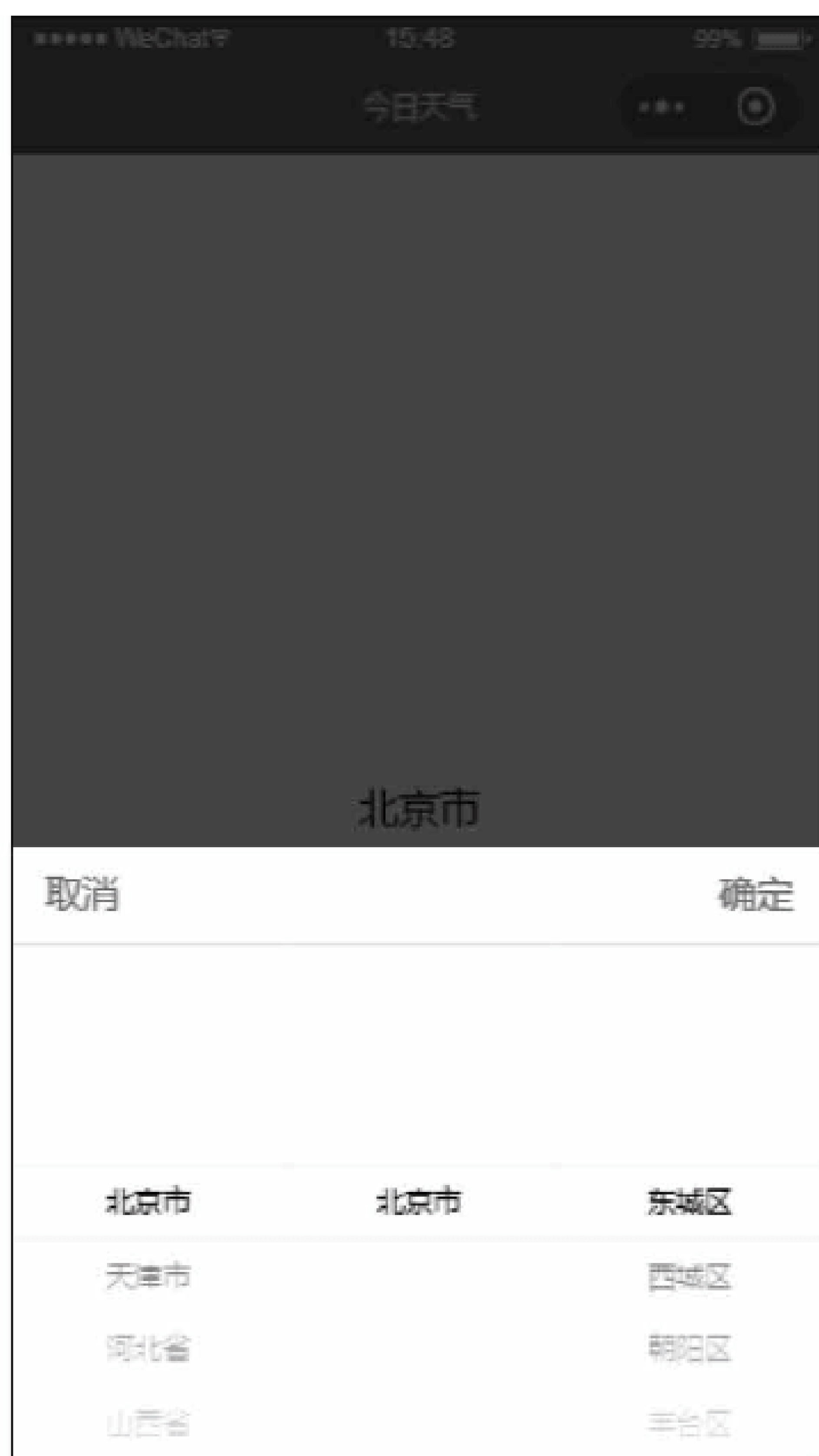
3 区域 2(文本)设计

区域 2 需要使用< text >组件实现一个单行天气信息,包括当前城市的温度和天气状况。

WXML(pages/index/index.wxml)代码片段修改如下:



(a) 页面初始效果



(b) 点击城市名称时的效果

图 5-13 区域 1 预览效果

```
1.  < view class = 'container'>
2.    <!-- 区域 1: 地区选择器 -->
3.    …
4.    <!-- 区域 2: 单行天气信息 -->
5.    < text > 19℃ 晴</text >
6.  </view>
```

WXSS(pages/index/index. wxss)代码片段如下：

```
1.  /* 文本样式 */
2.  text{
3.    font - size: 80rpx;
4.    color: #3C5F81;
5.  }
```

当前效果如图 5-14 所示。

当前显示的文本内容由开发者自定义，待查询到实际数据后将动态更新文本内容。

4 区域 3(天气图标)设计

区域 3 需要使用< image >组件展示当前城市的天气图标。

WXML(pages/index/index. wxml)代码片段修改如下：

```
1.  < view class = 'container'>
2.    <!-- 区域 1: 地区选择器 -->
3.    …
4.    <!-- 区域 2: 单行天气信息 -->
5.    …
6.    <!-- 区域 3: 天气图标 -->
7.    < image src = '/images/weather_icon/999.png' mode = 'widthFix'></image >
8.  </view>
```

WXSS(pages/index/index.wxss)代码片段如下：

```
1.  /* 图标样式 */
2.  image{
3.    width: 220rpx;
4.  }
```

当前效果如图 5-15 所示。



图 5-14　区域 2 预览效果



图 5-15　区域 3 预览效果

“N/A”表示天气状况为“未知”，待查询到实况数据后将动态更新图标内容。

5 区域 4(多行天气信息)设计

区域 4 需要使用< view >组件展示多行天气信息。

WXML(pages/index/index.wxml)代码片段修改如下：

```
1.  <view class = 'container'>
2.    <!-- 区域 1: 地区选择器 -->
3.    …
4.    <!-- 区域 2: 单行天气信息 -->
5.    …
6.    <!-- 区域 3: 天气图标 -->
7.    …
8.    <!-- 区域 4: 多行天气信息 -->
9.    <view class = 'detail'>
10.     <view class = 'bar'>
11.       <view class = 'box'>湿度</view>
12.       <view class = 'box'>气压</view>
13.       <view class = 'box'>能见度</view>
14.     </view>
15.     <view class = 'bar'>
16.       <view class = 'box'> 0 %</view>
17.       <view class = 'box'> 0 hPa</view>
```

```
18.         <view class = 'box'> 0 km </view>
19.       </view>
20.       <view class = 'bar'>
21.         <view class = 'box'>风向</view>
22.         <view class = 'box'>风速</view>
23.         <view class = 'box'>风力</view>
24.       </view>
25.       <view class = 'bar'>
26.         <view class = 'box'> 0 </view>
27.         <view class = 'box'> 0 km/h </view>
28.         <view class = 'box'> 0 级</view>
29.       </view>
30.     </view>
31. </view>
```

WXSS(pages/index/index.wxss)代码片段如下：

```
1.  /* 区域 4 整体样式 */
2.  .detail{
3.    width: 100%;
4.    display: flex;
5.    flex-direction: column;
6.  }
7.  /* 区域 4 单元行样式 */
8.  .bar{
9.    display: flex;
10.   flex-direction: row;
11.   margin: 20rpx 0;
12. }
13. /* 区域 4 单元格样式 */
14. .box{
15.   width: 33.3%;
16.   text-align: center;
17. }
```

当前效果如图 5-16 所示。

当前为开发者自定义数据，待查询到实况数据后将动态更新区域 4 的内容。此时页面设计就全部完成了，接下来需要进行逻辑实现。

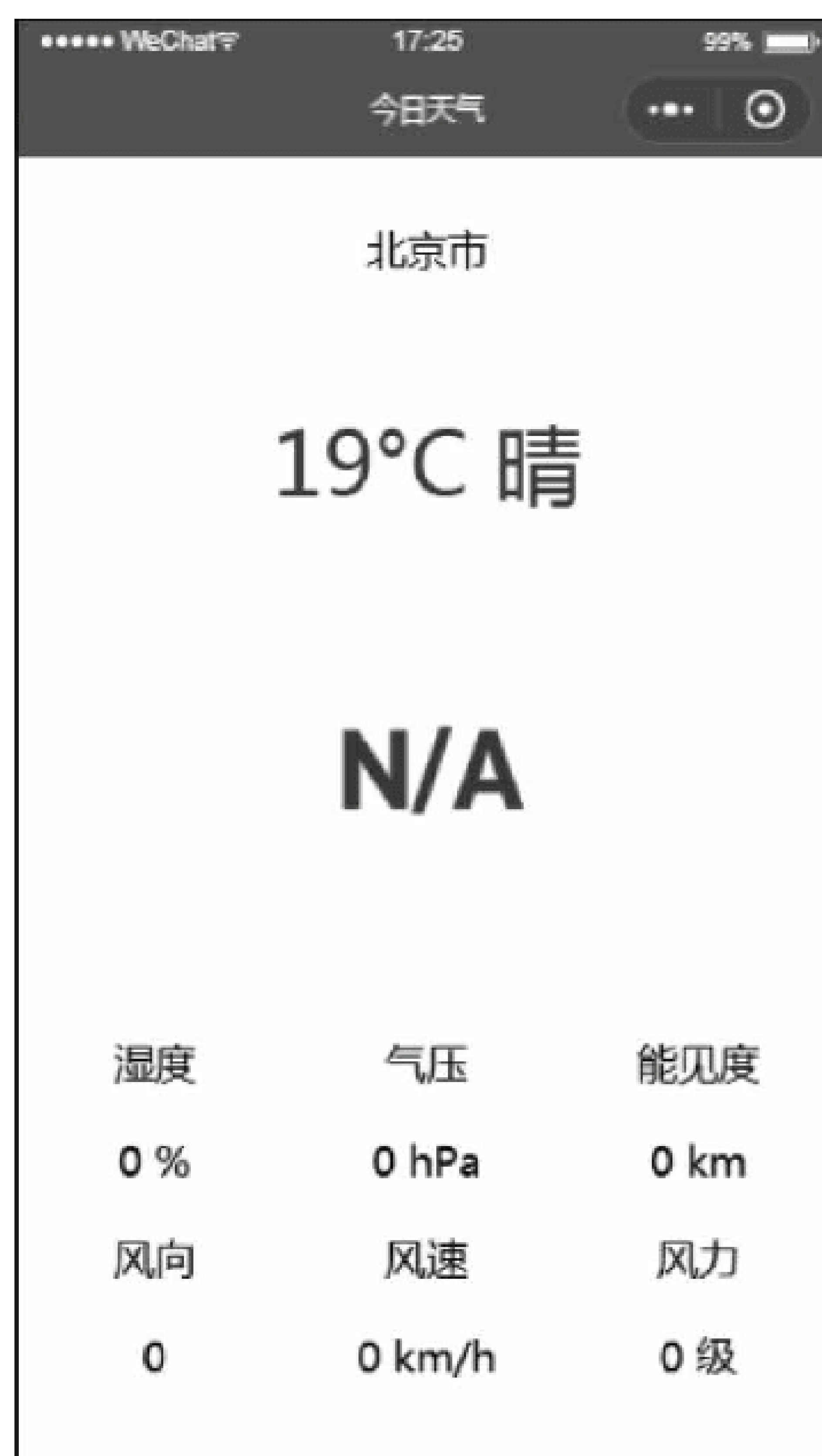
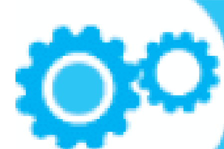


图 5-16 区域 4 预览效果



5.5 逻辑实现





视频讲解

5.5.1 更新省、市、区信息

首先修改<picker>组件中的“北京市”为{{region}}，然后为<picker>组件追加自定义 bindchange 事件，用于监听选项变化。

WXML(pages/index/index.wxml)代码片段修改如下：

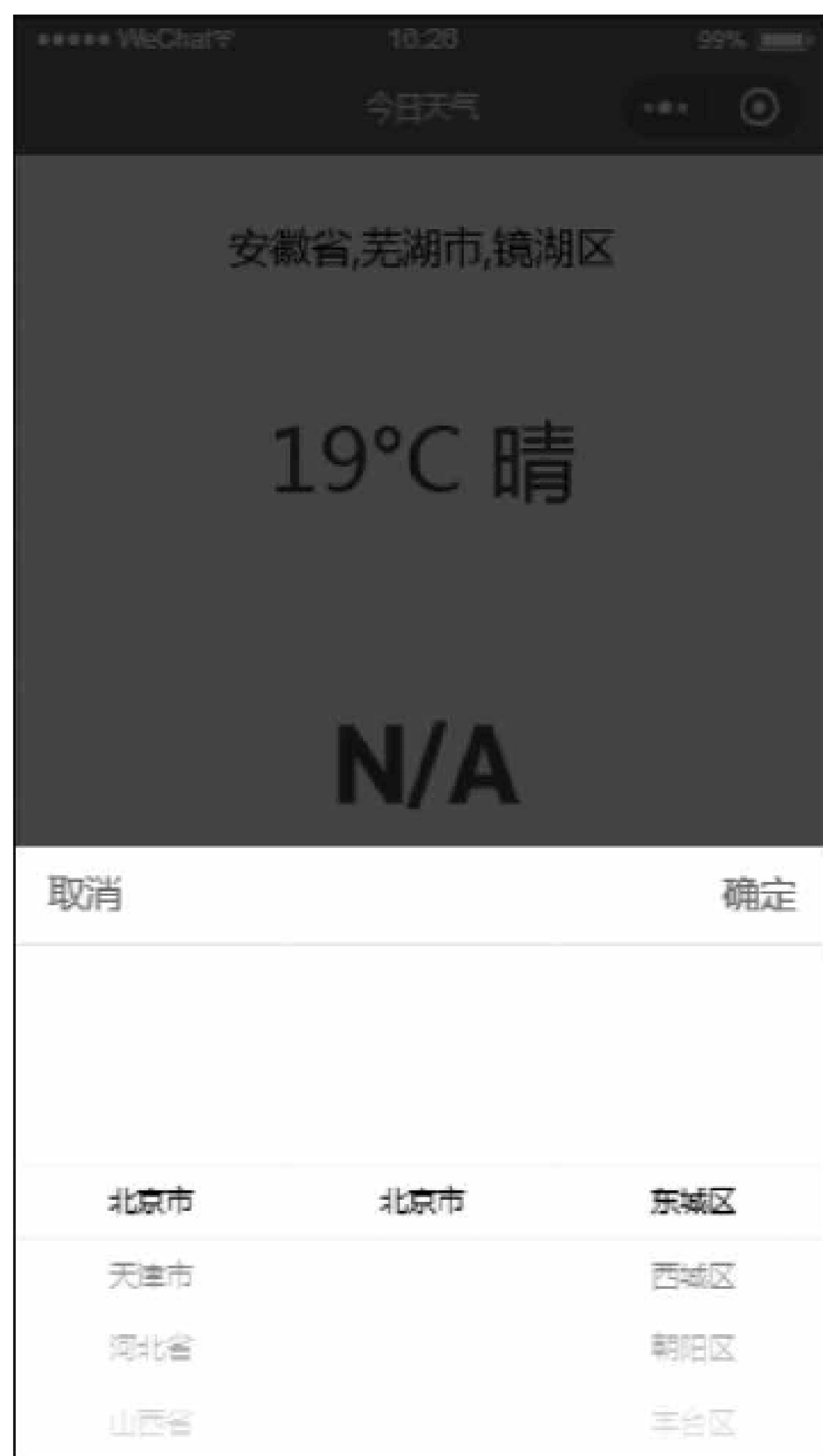
```
1. <view class = 'container'>
2.   <!-- 区域 1: 地区选择器 -->
3.   <picker mode = 'region' bindchange = 'regionChange'>
4.     <view>{{region}}</view>
5.   </picker>
6. </view>
```

由于地区选择器的返回结果是数组的形式，所以在 JS 文件的 data 中定义 region 为包含了省、市、区 3 个项目的数组，初始城市信息由开发者自定义。

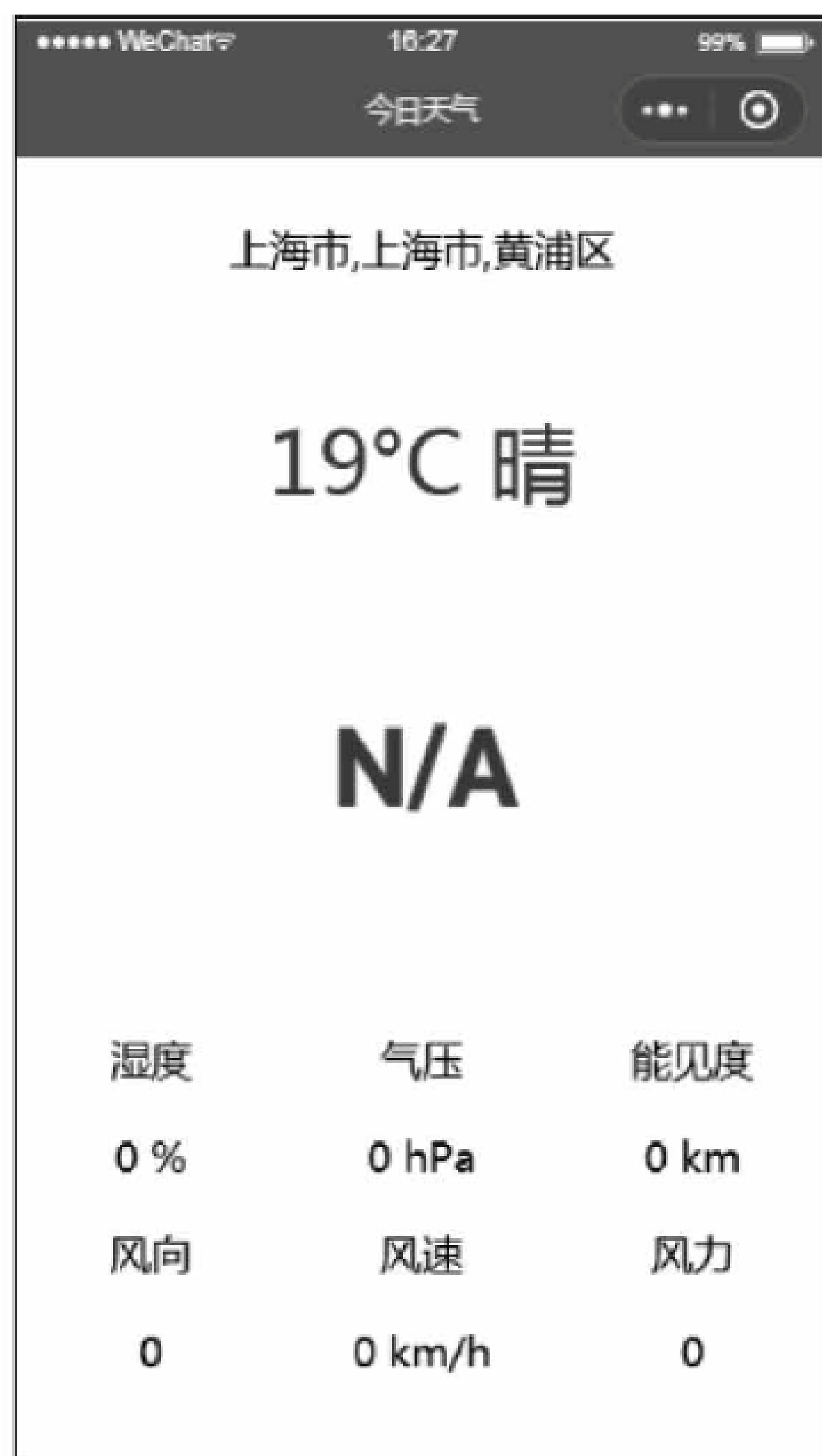
JS(pages/index/index.js)代码片段修改如下：

```
1.  Page({
2.    /**
3.     * 页面的初始数据
4.     */
5.    data: {
6.      region:['安徽省','芜湖市','镜湖区']
7.    },
8.    /**
9.     * 更新省、市、区信息
10.    */
11.   regionChange: function(e) {
12.     this.setData({region: e.detail.value});
13.   },
14. })
```

运行效果如图 5-17 所示。



(a) 重新选择城市



(b) 更新省、市、区信息

图 5-17　更新省、市、区信息前后

由图可见，当前已经可以自行切换到国内任意省、市、区。

5.5.2　获取实况天气数据



视频讲解

在 JS 文件中使用自定义函数 getWeather 进行实况天气数据的获取。由于非直辖市无法查询到具体的区，所以后续的天气查询以城市作为查询依据。

JS(pages/index/index.js)代码片段修改如下：

```
1.  Page({
2.    /**
3.     * 获取实况天气数据
4.     */
5.    getWeather: function() {
6.      var that = this;
7.      wx.request({
8.        url: 'https://free-api.heweather.com/s6/weather/now',
9.        data:{
10.         location:that.data.region[1],
11.         key:'请填入开发者申请的和风天气密钥'     //替换成开发者申请到的 key
12.       },
13.       success:function(res){
14.         console.log(res.data);
15.       }
16.     })
17.   },
18. })
```

将上述函数在生命周期函数 onLoad 和自定义函数 regionChange 中分别进行调用,表示当页面加载时和切换城市时均主动获取一次实况天气数据。

JS(pages/index/index.js)代码片段修改如下:

```
1.  Page({
2.    /**
3.     * 更新省、市、区信息
4.     */
5.    regionChange: function(e) {
6.      this.setData({region: e.detail.value});
7.      this.getWeather();                //更新天气
8.    },
9.    /**
10.    * 生命周期函数--监听页面加载
11.    */
12.   onLoad: function(options) {
13.     this.getWeather();                //更新天气
14.   },
15. })
```

在联网状态下保存后重新运行会在 Console 控制台得到第三方服务器发回的 JSON 数据,如图 5-18 所示。

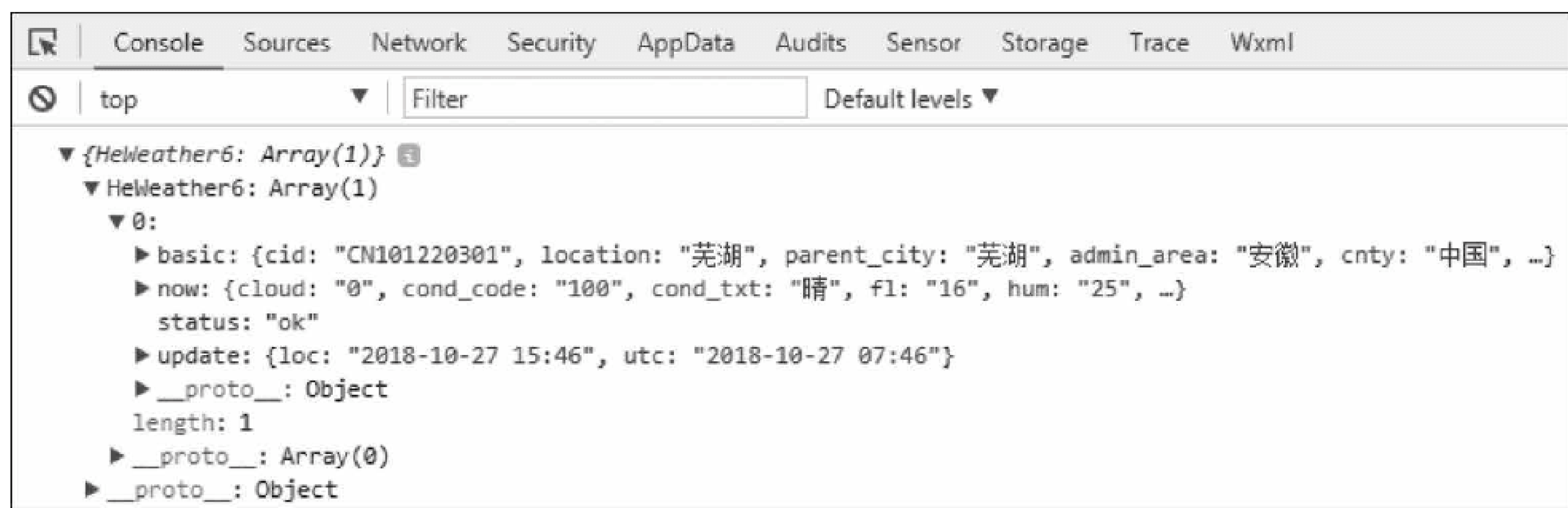


图 5-18 Console 控制台获取到服务器返回数据

由图可见，实况天气数据包含在 HeWeather6[0]. now 属性中。更新 getWeather 函数，将该属性存到 JS 文件的 data 中，JS(pages/index/index. js)代码片段修改如下：

```
1.  Page({
2.    /**
3.     * 获取实况天气数据
4.     */
5.    getWeather: function() {
6.      var that = this;
7.      wx.request({
8.        url: 'https://free-api.heweather.com/s6/
   weather/now',
9.        data:{
10.         location:that.data.region[1],
11.         key:'f0671b6589ff43019e72970d334ea93e'
12.       },
13.       success:function(res){
14.         that.setData({now:res.data.HeWeather6[0].now});
15.       }
16.     })
17.   },
18. })
```

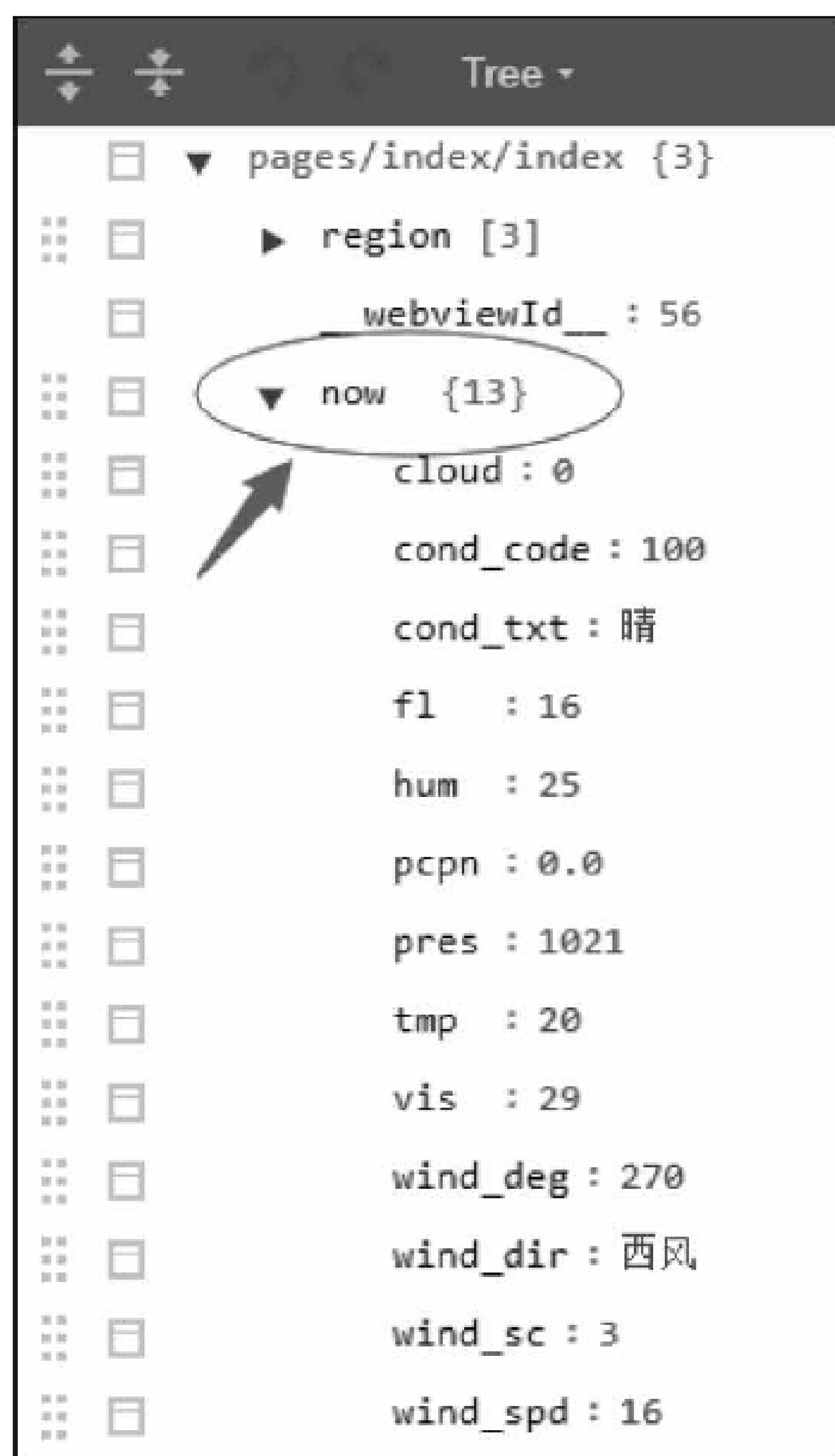


图 5-19 AppData 面板获取到数据

此时重新运行将在 AppData 面板中查到已经被记录的天气数据，如图 5-19 所示。

之后只需要将这些数据更新到页面上即可显示出来。

5.5.3 更新页面天气信息



视频讲解

将 WXML 页面上所有的临时数据都替换成{{now. 属性}}的形式，例如温度是{{now. tmp}}。

WXML(pages/index/index. wxml)代码片段修改如下：

```
1.  <view class='container'>
2.    <!-- 区域 1：地区选择器 -->
3.    …
4.    <!-- 区域 2：单行天气信息 -->
5.    <text>{{now.tmp}}℃ {{now.cond_txt}}</text>
6.    <!-- 区域 3：天气图标 -->
7.    <image src='/images/weather_icon/{{now.cond_code}}.png' mode='widthFix'></image>
8.    <!-- 区域 4：多行天气信息 -->
9.    <view class='detail'>
10.     <view class='bar'>
11.       <view class='box'>湿度</view>
12.       <view class='box'>气压</view>
13.       <view class='box'>能见度</view>
14.     </view>
15.     <view class='bar'>
16.       <view class='box'>{{now.hum}} %</view>
17.       <view class='box'>{{now.pres}} hPa</view>
18.       <view class='box'>{{now.vis}} km</view>
19.     </view>
20.     <view class='bar'>
21.       <view class='box'>风向</view>
22.       <view class='box'>风速</view>
```

```
23.         <view class='box'>风力</view>
24.       </view>
25.       <view class='bar'>
26.         <view class='box'>{{now.wind_dir}}</view>
27.         <view class='box'>{{now.wind_spd}} km/h</view>
28.         <view class='box'>{{now.wind_sc}} 级</view>
29.       </view>
30.     </view>
31. </view>
```

运行效果如图 5-20 所示。

需要注意的是，在网速受限的情况下可能不能立刻获取到数据，因此最好在 JS 文件的 data 中为 now 规定初始数据，在获取到实际数据前可以临时显示这些数据。

JS(pages/index/index.js)代码片段修改如下：

```
1.  Page({
2.    /**
3.     * 页面的初始数据
4.     */
5.    data: {
6.      region: ['安徽省', '芜湖市', '镜湖区'],
7.      now:{
8.        tmp:0,
9.        cond_txt:'未知',
10.       cond_code:'999',
11.       hum:0,
12.       pres:0,
13.       vis:0,
14.       wind_dir:0,
15.       wind_spd:0,
16.       wind_sc:0
17.     }
18.   },
19. })
```

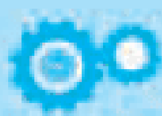
在网速受限的状态下，初始数据显示效果如图 5-21 所示。



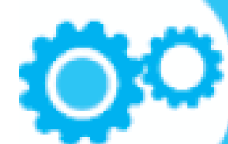
图 5-20 实况天气数据显示效果



图 5-21 初始数据显示效果



此时项目就全部完成了。



5.6 完整代码展示



app.json 文件的完整代码如下：

```
1.  {
2.    "pages": [
3.      "pages/index/index"
4.    ],
5.    "window": {
6.      "navigationBarBackgroundColor": "#3883FA",
7.      "navigationBarTitleText": "今日天气"
8.    }
9.  }
```

app.wxss 文件的完整代码如下：

```
1.  /* 背景容器样式 */
2.  .container{
3.    height: 100vh;                       /* 高度为 100 视窗,写成 100% 无效 */
4.    display: flex;                       /* flex 布局模型 */
5.    flex-direction: column;              /* 垂直布局 */
6.    align-items: center;                 /* 水平方向居中 */
7.    justify-content: space-around;       /* 调整内容位置 */
8.  }
```

WXML 文件(pages/index/index.wxml)的完整代码如下：

```
1.  <!-- index.wxml -->
2.  <view class='container'>
3.    <!-- 区域 1: 地区选择器 -->
4.    <picker mode='region' bindchange='regionChange'>
5.      <view>{{region}}</view>
6.    </picker>
7.
8.    <!-- 区域 2: 单行天气信息 -->
9.    <text>{{now.tmp}}°C {{now.cond_txt}}</text>
10.
11.   <!-- 区域 3: 天气图标 -->
12.   <image src='/images/weather_icon/{{now.cond_code}}.png' mode='widthFix'></image>
13.
14.   <!-- 区域 4: 多行天气信息 -->
15.   <view class='detail'>
16.     <view class='bar'>
17.       <view class='box'>湿度</view>
18.       <view class='box'>气压</view>
19.       <view class='box'>能见度</view>
20.     </view>
21.     <view class='bar'>
22.       <view class='box'>{{now.hum}} %</view>
23.       <view class='box'>{{now.pres}} hPa</view>
24.       <view class='box'>{{now.vis}} km</view>
25.     </view>
```

```
26.       <view class='bar'>
27.         <view class='box'>风向</view>
28.         <view class='box'>风速</view>
29.         <view class='box'>风力</view>
30.       </view>
31.       <view class='bar'>
32.         <view class='box'>{{now.wind_dir}}</view>
33.         <view class='box'>{{now.wind_spd}} km/h</view>
34.         <view class='box'>{{now.wind_sc}} 级</view>
35.       </view>
36.     </view>
37.   </view>
```

WXSS 文件(pages/index/index.wxss)的完整代码如下：

```
1.  /* 文本样式 */
2.  text{
3.    font-size: 80rpx;
4.    color: #3C5F81;
5.  }
6.
7.  /* 图标样式 */
8.  image{
9.    width: 220rpx;
10. }
11.
12. /* 区域 4 整体样式 */
13. .detail{
14.   width: 100%;
15.   display: flex;
16.   flex-direction: column;
17. }
18. /* 区域 4 单元行样式 */
19. .bar{
20.   display: flex;
21.   flex-direction: row;
22.   margin: 20rpx 0;
23. }
24. /* 区域 4 单元格样式 */
25. .box{
26.   width: 33.3%;
27.   text-align: center;
28. }
```

JS 文件(pages/index/index.js)的完整代码如下：

```
1.  Page({
2.    /**
3.     * 页面的初始数据
4.     */
5.    data: {
6.      region: ['安徽省', '芜湖市', '镜湖区'],
7.      now:{
8.        tmp:0,
9.        cond_txt:'未知',
10.       cond_code:'999',
```

```
11.         hum:0,
12.         pres:0,
13.         vis:0,
14.         wind_dir:0,
15.         wind_spd:0,
16.         wind_sc:0
17.       }
18.     },
19.     /**
20.      * 更新省、市、区信息
21.      */
22.     regionChange: function(e) {
23.       this.setData({region: e.detail.value});
24.       this.getWeather();              //更新天气
25.     },
26.     /**
27.      * 获取实况天气数据
28.      */
29.     getWeather: function() {
30.       var that = this;
31.       wx.request({
32.         url: 'https://free-api.heweather.com/s6/weather/now',
33.         data:{
34.           location:that.data.region[1],
35.           key:'请填入开发者申请的和风天气密钥'     //替换成开发者申请到的 key
36.         },
37.         success:function(res){
38.           that.setData({now:res.data.HeWeather6[0].now});
39.         }
40.       })
41.     },
42.     /**
43.      * 生命周期函数--监听页面加载
44.      */
45.     onLoad: function(options) {
46.       this.getWeather();              //更新天气
47.     }
48.   })
```

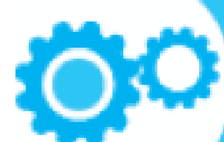
第6章  Chapter 6

小程序媒体API · 口述校史

本项目主要内容是使用小程序媒体API制作一个视频播放小程序，视频素材来源为安徽师范大学档案馆的《口述校史》栏目，它录制了多名耄耋之年的老教工回忆工作时期对大学的印象。

本章学习目标

- 掌握视频列表的切换方法；
- 掌握视频自动播放方法；
- 掌握视频随机颜色弹幕效果。



6.1 项目创建



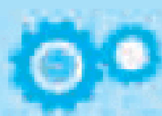


视频讲解

本项目创建选择空白文件夹videoDemo，效果如图6-1所示。

单击“新建”按钮完成项目创建，然后准备手动修改页面配置文件。

图6-1 小程序项目填写效果示意图





6.2 页面配置





6.2

6.2.1 创建页面文件

项目创建完毕后，在根目录中会生成文件夹 pages 用于存放页面文件。一般来说首页默认命名为 index，表示小程序运行的第一个页面；其他页面名称可以自定义。本项目只需要保留首页(index)即可。

具体操作如下：

（1）将 app.json 文件内 pages 属性中的“pages/logs/logs”删除，并删除上一行末尾的逗号。

（2）按快捷键 Ctrl+S 保存当前修改。

6.2.2 删除和修改文件

具体操作如下：

（1）删除 utils 文件夹及其内部所有内容。

（2）删除 pages 文件夹下的 logs 目录及其内部所有内容。

（3）删除 index.wxml 和 index.wxss 中的全部代码。

（4）删除 index.js 中的全部代码，并且输入关键词 page 找到第二个选项按回车键让其自动补全函数（如图 6-2 所示）。



图 6-2　输入关键词创建 Page 函数

（5）删除 app.wxss 中的全部代码。

（6）删除 app.js 中的全部代码，并且输入关键词 app 找到第一个选项按回车键让其自动补全函数（如图 6-3 所示）。

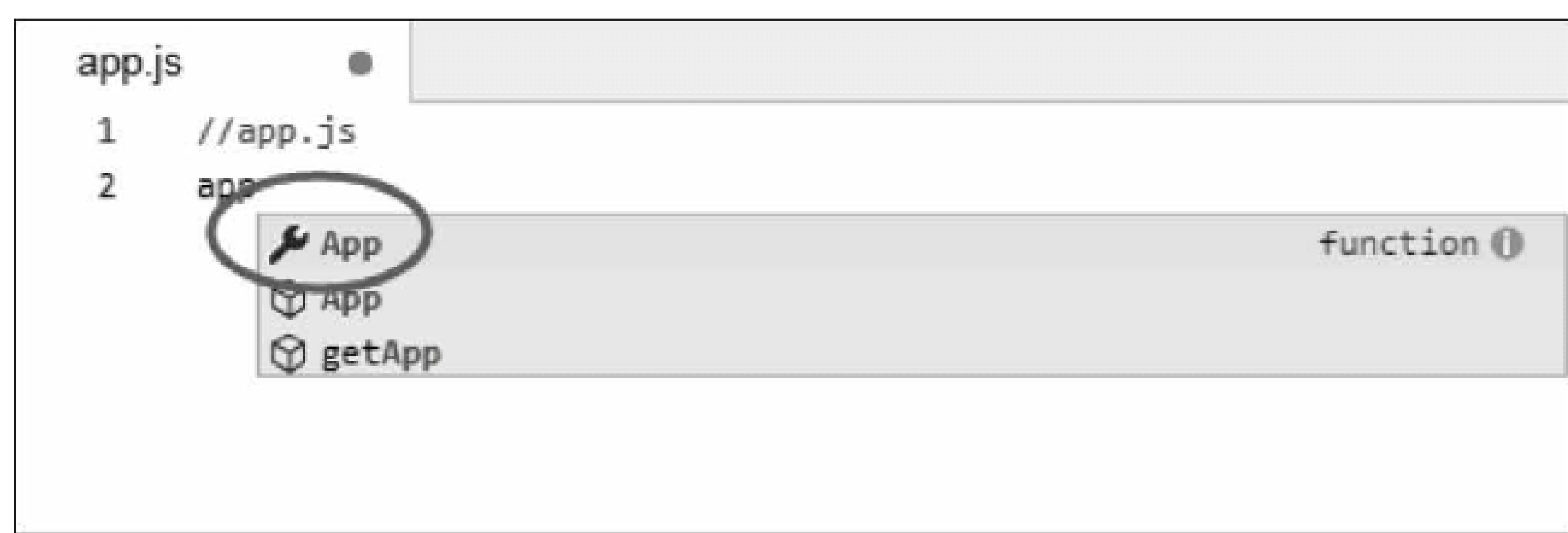


图 6-3　输入关键词创建 App 函数

6.2.3 创建其他文件

接下来创建其他自定义文件,本项目还需要一个文件夹用于存放播放图标。文件夹名称由开发者自定义(例如 images),单击目录结构左上角的+号创建文件夹并命名为 images。

本项目用到的图标素材如图 6-4 所示。

右击目录结构中的 images 文件夹,选择"硬盘打开",然后将图标文件复制、粘贴进该目录。完成后的目录结构如图 6-5 所示。

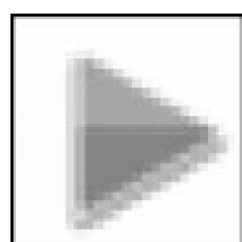
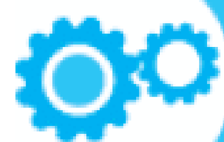


图 6-4 图标素材展示



图 6-5 页面文件创建完成

此时文件配置就全部完成,6.3 节将正式进行页面布局和样式设计。

 6.3 视图设计 

6.3.1 导航栏设计



视频讲解

小程序默认导航栏是黑底白字的效果,因此需要在 app.json 中自定义导航栏标题和背景颜色。更改后的 app.json 文件代码如下:

```
1.  {
2.    "pages": [
3.      "pages/index/index"
4.    ],
5.    "window": {
6.      "navigationBarBackgroundColor": "#987938",
7.      "navigationBarTitleText": "口述校史"
8.    }
9.  }
```



图 6-6 自定义导航栏效果

上述代码可以更改所有页面的导航栏标题文本为"口述校史"、背景颜色为金棕色(#987938)。预览效果如图 6-6 所示。

6.3.2 页面设计



视频讲解

页面上主要包含 3 个区域,具体内容解释如下。

- 区域 1：视频播放器，用于播放指定的视频；
- 区域 2：弹幕发送区域，包含文本输入框和发送按钮；
- 区域 3：视频列表，垂直排列多个视频标题，点击不同的标题播放对应的视频内容。

面板之间需要有一定的间隔距离，设计图如图 6-7 所示。



图 6-7　页面设计图

计划使用如下组件。

- 区域 1：< video >组件；
- 区域 2：< view >组件，并定义 class= 'danmuArea'；
- 区域 2 内部：< input >和< button >组件；
- 区域 3：< view >组件，并定义 class= 'videoList'；
- 区域 3 内单元行：< view >组件，并定义 class= 'videoBar'；
- 区域 3 单元行内：每行一个< image >组件用于显示播放图标、一个< text >组件用于显示视频标题。

1 区域 1（视频组件）设计

区域 1 需要使用< video >组件来实现一个视频播放器。

WXML（pages/index/index. wxml）代码片段如下：

```
1.  <!-- 区域 1：视频播放器 -->
2.  < video id = 'myVideo' controls ></video >
```

其中 controls 属性用于显示播放/暂停、音量等控制组件。

WXSS（pages/index/index. wxss）代码片段如下：

```
1.  /* 视频组件样式 */
2.  video {
3.    width: 100%;                /* 视频组件宽度为 100% */
4.  }
```

当前效果如图 6-8 所示。

由图可见，此时视频播放区域已经出现在屏幕的顶端。

2 区域 2（弹幕区域）设计

区域 2 需要使用< view >组件实现一个单行区域，包括文本输入框和发送按钮。

WXML（pages/index/index. wxml）代码片段修改如下：

```
1.  <!-- 区域 1：视频播放器 -->
2.    …
3.  <!-- 区域 2：弹幕控制 -->
4.  < view class = 'danmuArea'>
5.    < input type = 'text' placeholder = '请输入弹幕内容'></input >
6.    < button >发送弹幕</button >
7.  </view >
```

WXSS（pages/index/index. wxss）代码片段如下：

```
1.  /* 区域 2：弹幕控制样式 */
2.  /* 2-1 弹幕区域样式 */
3.  .danmuArea {
4.    display: flex;              /* flex 模型布局 */
```

```
5.     flex - direction: row;              /* 水平方向排列 */
6.   }
7.   /* 2 - 2 文本输入框样式 */
8.   input {
9.     border: 1rpx solid #987938;         /* 1rpx 宽的实线棕色边框 */
10.    flex - grow: 1;                     /* 扩张多余空间宽度 */
11.    height: 100rpx;                     /* 高度 */
12.  }
13.  /* 2 - 3 按钮样式 */
14.  button {
15.    color: white;                       /* 字体颜色 */
16.    background - color: #987938;        /* 背景颜色 */
17.  }
```

当前效果如图 6-9 所示。

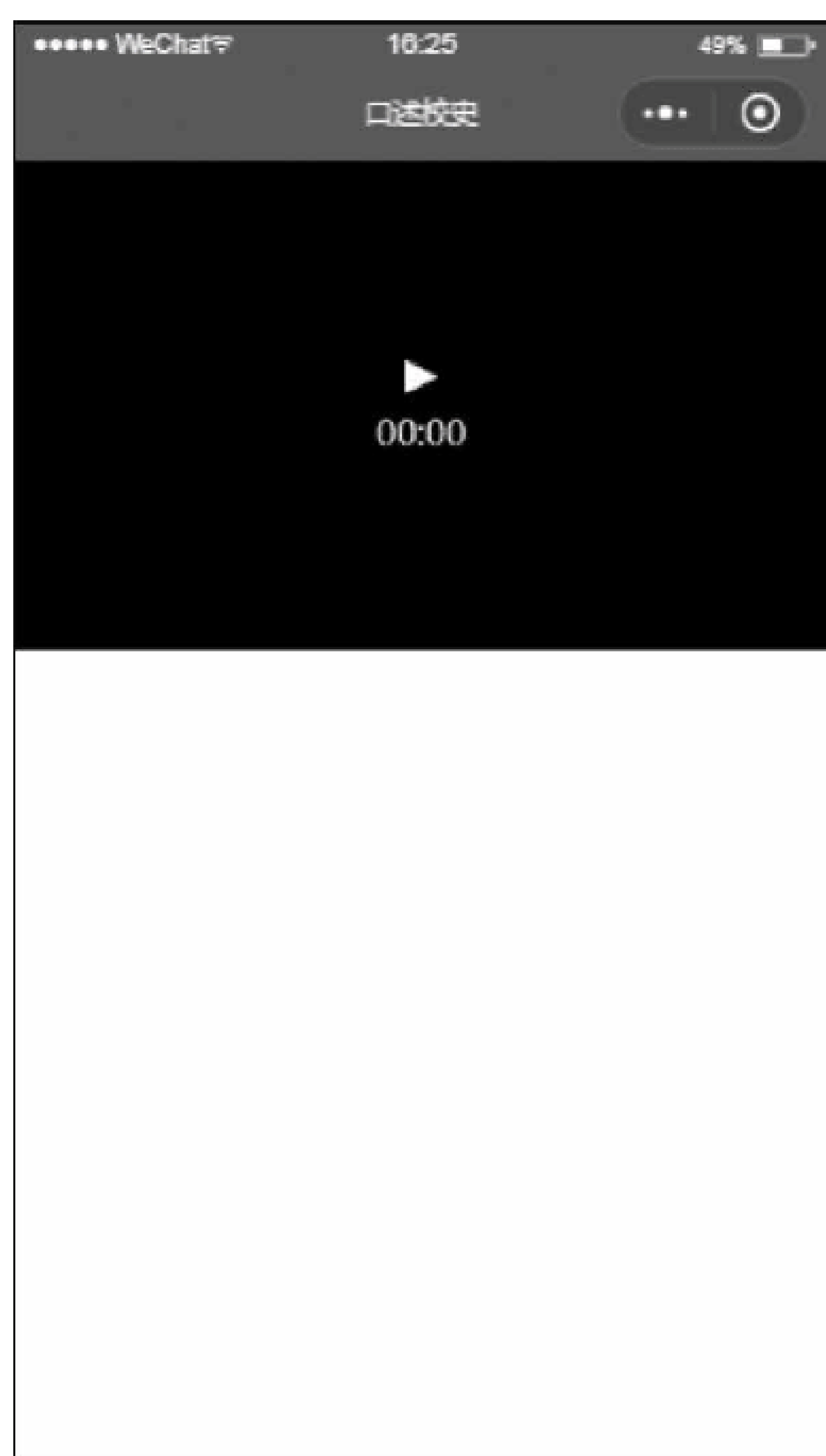


图 6-8　区域 1 预览效果



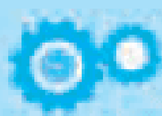
图 6-9　区域 2 预览效果

3 区域 3(视频列表)设计

区域 3 需要使用< view >组件实现一个可扩展的多行区域,每行包含一个播放图标和一个视频标题文本。当前先设计第一行效果,后续使用 wx:for 属性循环添加全部内容。

WXML(pages/index/index.wxml)代码片段修改如下:

```
1.  <!-- 区域 1: 视频播放器 -->
2.    …
3.  <!-- 区域 2: 弹幕控制 -->
4.    …
5.  <!-- 区域 3: 视频列表 -->
6.  <view class = 'videoList'>
7.    <view class = 'videoBar'>
8.      <image src = '/images/play.png'></image>
```



```
9.        <text>这是一个测试标题</text>
10.     </view>
11.  </view>
```

WXSS(pages/index/index.wxss)代码片段如下：

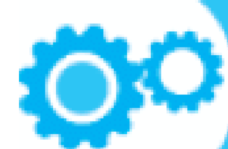
```
1.  /* 区域 3: 视频列表样式 */
2.  /* 3-1 视频列表区域样式 */
3.  .videoList {
4.    width: 100%;                          /* 宽度 */
5.    min-height: 400rpx;                   /* 最小高度 */
6.  }
7.  /* 3-2 单行列表区域样式 */
8.  .videoBar {
9.    width: 95%;                           /* 宽度 */
10.   display: flex;                        /* flex 模型布局 */
11.   flex-direction: row;                  /* 水平方向布局 */
12.   border-bottom: 1rpx solid #987938;    /* 1rpx 宽的实线棕色边框 */
13.   margin: 10rpx;                        /* 外边距 */
14. }
15. /* 3-3 播放图标样式 */
16. image {
17.   width: 70rpx;                         /* 宽度 */
18.   height: 70rpx;                        /* 高度 */
19.   margin: 20rpx;                        /* 外边距 */
20. }
21. /* 3-4 文本标题样式 */
22. text {
23.   font-size: 45rpx;                     /* 字体大小 */
24.   color: #987938;                       /* 字体颜色为棕色 */
25.   margin: 20rpx;                        /* 外边距 */
26.   flex-grow: 1;                         /* 扩张多余空间宽度 */
27. }
```

当前效果如图 6-10 所示。



图 6-10　区域 3 预览效果

此时页面设计就全部完成了，接下来需要进行逻辑实现。



6.4 逻辑实现



6.4.1 更新播放列表



视频讲解

在区域 3 对< view class='videoBar'>组件添加 wx:for 属性，改写为循环展示列表。

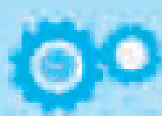
WXML(pages/index/index.wxml)代码片段修改如下：

```
1.  <! -- 区域 1:视频播放器 -->
2.  < video id = "myVideo" controls src = "{{src}}"></video >
3.  <! -- 区域 2:弹幕控制区域(代码略) -->
4.  <! -- 区域 3:视频列表 -->
5.  < view class = 'videoList'>
6.    < view class = 'videoBar' wx:for = '{{list}}' wx:key = 'video{{index}}'>
7.      < image src = '/images/play.png'></image >
8.      < text >{{item.title}}</text >
9.    </view >
10. </view >
```

然后在 JS 文件的 data 属性中追加 list 数组，用于存放视频信息。

JS(pages/index/index.js)代码片段修改如下：

```
1.  Page({
2.    /**
3.     * 页面的初始数据
4.     */
5.    data: {
6.      list: [
7.       {
8.           id: '299371',
9.           title: '杨国宜先生口述校史实录',
10.          videoUrl: 'http://arch.ahnu.edu.cn/__local/6/CB/D1/C2DF3FC847F4CE2ABB67034C595_
025F0082_ABD7AE2.mp4?e = .mp4'
11.        },
12.        {
13.          id: '299396',
14.          title: '唐成伦先生口述校史实录',
15.          videoUrl: 'http://arch.ahnu.edu.cn/__local/E/31/EB/2F368A265E6C842BB6A63EE5F97_
425ABEDD_7167F22.mp4?e = .mp4'
16.        },
17.        {
18.          id: '299378',
19.          title: '倪光明先生口述校史实录',
20.          videoUrl: 'http://arch.ahnu.edu.cn/__local/9/DC/3B/35687573BA2145023FDAEBAFE67_
AAD8D222_925F3FF.mp4?e = .mp4'
21.        },
22.        {
23.          id: '299392',
24.          title: '吴兴仪先生口述校史实录',
25.          videoUrl: 'http://arch.ahnu.edu.cn/__local/5/DA/BD/7A27865731CF2B096E90B522005_
A29CB142_6525BCF.mp4?e = .mp4'
26.        }
```

```
27.         ]
28.     },
29.  })
```

运行效果如图 6-11 所示。



图 6-11 更新视频列表效果

由图可见，当前已经可以展示全部视频列表。

6.4.2 点击播放视频



视频讲解

在区域 3 对< view class＝'videoBar'>组件添加 data-url 属性和 bindtap 属性。其中 data-url 用于记录每行视频对应的播放地址，bindtap 用于触发点击事件。

WXML(pages/index/index.wxml)代码片段修改如下：

```
1.  <!-- 区域 3: 视频列表 -->
2.  <view class = 'videoList'>
3.    <view class = 'videoBar' wx:for = '{{list}}' wx:key = 'video{{index}}' data - url = '{{item.
videoUrl}}' bindtap = 'playVideo'>
4.      <image src = '/images/play.png'></image>
5.      <text>{{item.title}}</text>
6.    </view>
7.  </view>
```

然后在 JS 文件的 onLoad 函数中创建视频上下文，用于控制视频的播放和停止。

JS(pages/index/index.js)代码片段修改如下：

```
1.  Page({
2.    /**
3.     * 生命周期函数 -- 监听页面加载
4.     */
5.    onLoad: function(options) {
```

```
6.        this.videoCtx = wx.createVideoContext('myVideo')
7.      },
8.    })
```

接着添加自定义函数 playVideo,JS(pages/index/index.js)代码片段如下:

```
1.   Page({
2.     /**
3.      * 播放视频
4.      */
5.     playVideo: function(e) {
6.       //停止之前正在播放的视频
7.       this.videoCtx.stop()
8.       //更新视频地址
9.       this.setData({
10.        src: e.currentTarget.dataset.url
11.      })
12.      //播放新的视频
13.      this.videoCtx.play()
14.    },
15.  })
```

运行效果如图 6-12 所示。



(a) 页面初始效果



(b) 点击播放视频

图 6-12　点击播放视频效果

由图可见,当前已经可以成功播放视频列表中的视频。

6.4.3　发送弹幕



视频讲解

在区域 1 对< video >组件添加 enable-danmu 和 danmu-btn 属性,用于允许发送弹幕和显示“发送弹幕”按钮。

WXML(pages/index/index.wxml)代码片段修改如下：

```
1.  <!-- 区域 1: 视频播放器 -->
2.  <video id='myVideo' src='{{src}}' controls enable-danmu danmu-btn></video>
```

然后在区域 2 为文本输入框追加 bindinput 属性，用于获取弹幕文本内容；为按钮追加 bindtap 属性，用于触发点击事件。

WXML(pages/index/index.wxml)代码片段修改如下：

```
1.  <!-- 区域 2: 弹幕控制 -->
2.  <view class='danmuArea'>
3.    <input type='text' placeholder='请输入弹幕内容' bindinput='getDanmu'></input>
4.    <button bindtap='sendDanmu'>发送弹幕</button>
5.  </view>
```

对应的 JS(pages/index/index.js)代码片段修改如下：

```
1.  Page({
2.    /**
3.     * 页面的初始数据
4.     */
5.    data: {
6.      danmuTxt: '',
7.      list: [ … ]
8.    }
9.    /**
10.    * 更新弹幕内容
11.    */
12.   getDanmu: function(e) {
13.     this.setData({
14.       danmuTxt: e.detail.value
15.     })
16.   },
17.   /**
18.    * 发送弹幕
19.    */
20.   sendDanmu: function(e) {
21.     let text = this.data.danmuTxt;
22.     this.videoCtx.sendDanmu({
23.       text:text,
24.       color: 'red'
25.     })
26.   },
27. })
```

此时可以发出红色文本的弹幕，运行效果如图 6-13 所示。

如果希望发出随机颜色的弹幕内容，可以在 JS 文件中追加自定义函数 getRandomColor。JS(pages/index/index.js)代码片段如下：

```
1.  //生成随机颜色
2.  function getRandomColor() {
3.    let rgb = []
4.    for (let i = 0; i < 3; ++i) {
5.      let color = Math.floor(Math.random() * 256).toString(16)
6.      color = color.length == 1 ? '0' + color : color
```

```
7.         rgb.push(color)
8.       }
9.     return '#' + rgb.join('')
10. }
```

上述代码可以随机生成一个十六进制的颜色，将其在原先需要录入 color 属性的地方调用即可实现彩色弹幕效果。JS(pages/index/index.js)代码片段修改如下：

```
1.   Page({
2.     /**
3.      * 发送弹幕
4.      */
5.     sendDanmu: function(e) {
6.       let text = this.data.danmuTxt;
7.       this.videoCtx.sendDanmu({
8.         text:text,
9.         color: getRandomColor()
10.       })
11.     },
12. })
```

此时可以发出彩色文本的弹幕，运行效果如图 6-14 所示。



图 6-13　发送红色弹幕效果



图 6-14　发送彩色弹幕效果

至此相关的逻辑功能均已实现，项目全部完成。



6.5　完整代码展示



app.json 文件的完整代码如下：

```
1.   {
```

```
2.    "pages": [
3.      "pages/index/index"
4.    ],
5.    "window": {
6.      "navigationBarBackgroundColor": "#987938",
7.      "navigationBarTitleText": "口述校史"
8.    }
9.  }
```

WXML 文件(pages/index/index.wxml)的完整代码如下：

```
1.  <!-- index.wxml -->
2.  <!-- 区域 1：视频播放器 -->
3.  <video id='myVideo' src='{{src}}' controls enable-danmu danmu-btn></video>
4.  <!-- 区域 2：弹幕控制 -->
5.  <view class='danmuArea'>
6.    <input type='text' placeholder='请输入弹幕内容' bindinput='getDanmu'></input>
7.    <button bindtap='sendDanmu'>发送弹幕</button>
8.  </view>
9.  <!-- 区域 3：视频列表 -->
10. <view class='videoList'>
11.   <view class='videoBar' wx:for='{{list}}' wx:key='video{{index}}' data-url='{{item.
videoUrl}}' bindtap='playVideo'>
12.     <image src='/images/play.png'></image>
13.     <text>{{item.title}}</text>
14.   </view>
15. </view>
```

WXSS 文件(pages/index/index.wxss)的完整代码如下：

```
1.  /* 区域 1：视频组件样式 */
2.  video {
3.    width: 100%;                      /* 视频组件宽度为 100% */
4.  }
5.
6.  /* 区域 2：弹幕控制样式 */
7.  /* 2-1 弹幕区域样式 */
8.  .danmuArea {
9.    display: flex;                    /* flex 模型布局 */
10.   flex-direction: row;              /* 水平方向排列 */
11. }
12. /* 2-2 文本输入框样式 */
13. input {
14.   border: 1rpx solid #987938;       /* 1rpx 宽的实线棕色边框 */
15.   flex-grow: 1;                     /* 扩张多余空间宽度 */
16.   height: 100rpx;                   /* 高度 */
17. }
18. /* 2-3 按钮样式 */
19. button {
20.   color: white;                     /* 字体颜色 */
21.   background-color: #987938;        /* 背景颜色 */
22. }
23.
24. /* 区域 3：视频列表样式 */
25. /* 3-1 视频列表区域样式 */
26. .videoList {
```

```
27.    width: 100 %;                          /* 宽度 */
28.    min-height: 400rpx;                    /* 最小高度 */
29.  }
30.  /* 3-2 单行列表区域样式 */
31.  .videoBar {
32.    width: 95 %;                           /* 宽度 */
33.    display: flex;                         /* flex 模型布局 */
34.    flex-direction: row;                   /* 水平方向布局 */
35.    border-bottom: 1rpx solid #987938;     /* 1rpx 宽的实线棕色边框 */
36.    margin: 10rpx;                         /* 外边距 */
37.  }
38.  /* 3-3 播放图标样式 */
39.  image {
40.    width: 70rpx;                          /* 宽度 */
41.    height: 70rpx;                         /* 高度 */
42.    margin: 20rpx;                         /* 外边距 */
43.  }
44.  /* 3-4 文本标题样式 */
45.  text {
46.    font-size: 45rpx;                      /* 字体大小 */
47.    color: #987938;                        /* 字体颜色为棕色 */
48.    margin: 20rpx;                         /* 外边距 */
49.    flex-grow: 1;                          /* 扩张多余空间宽度 */
50.  }
```

JS(pages/index/index.js)的完整代码如下：

```
1.   //生成随机颜色
2.   function getRandomColor() {
3.     let rgb = []
4.     for (let i = 0; i < 3; ++i) {
5.       let color = Math.floor(Math.random() * 256).toString(16)
6.       color = color.length == 1 ? '0' + color : color
7.       rgb.push(color)
8.     }
9.     return '#' + rgb.join('')
10.  }
11.
12.  Page({
13.    /**
14.     * 页面的初始数据
15.     */
16.    data: {
17.      danmuTxt: '',
18.      list: [{
19.          id: '299371',
20.          title: '杨国宜先生口述校史实录',
21.          videoUrl: 'http://arch.ahnu.edu.cn/__local/6/CB/D1/C2DF3FC847F4CE2ABB67034C595_
025F0082_ABD7AE2.mp4?e=.mp4'
22.        },
23.        {
24.          id: '299396',
25.          title: '唐成伦先生口述校史实录',
26.          videoUrl: 'http://arch.ahnu.edu.cn/__local/E/31/EB/2F368A265E6C842BB6A63EE5F97_
425ABEDD_7167F22.mp4?e=.mp4'
27.        },
```

```
28.          {
29.            id: '299378',
30.            title: '倪光明先生口述校史实录',
31.            videoUrl: 'http://arch.ahnu.edu.cn/__local/9/DC/3B/35687573BA2145023FDAEBAFE67_
AAD8D222_925F3FF.mp4?e=.mp4'
32.          },
33.          {
34.            id: '299392',
35.            title: '吴兴仪先生口述校史实录',
36.            videoUrl: 'http://arch.ahnu.edu.cn/__local/5/DA/BD/7A27865731CF2B096E90B522005_
A29CB142_6525BCF.mp4?e=.mp4'
37.          }
38.        ]
39.    },
40.    /**
41.      * 更新弹幕内容
42.      */
43.    getDanmu: function(e) {
44.      this.setData({
45.        danmuTxt: e.detail.value
46.      })
47.    },
48.    /**
49.      * 发送弹幕
50.      */
51.    sendDanmu: function(e) {
52.      let text = this.data.danmuTxt;
53.      this.videoCtx.sendDanmu({
54.        text:text,
55.        color: getRandomColor()
56.      })
57.    },
58.    /**
59.      * 播放视频
60.      */
61.    playVideo: function(e) {
62.      //停止之前正在播放的视频
63.      this.videoCtx.stop()
64.      //更新视频地址
65.      this.setData({
66.        src: e.currentTarget.dataset.url
67.      })
68.      //播放新的视频
69.      this.videoCtx.play()
70.    },
71.    /**
72.      * 生命周期函数--监听页面加载
73.      */
74.    onLoad: function(options) {
75.      this.videoCtx = wx.createVideoContext('myVideo')
76.    }
77. })
```

第7章 Chapter 7

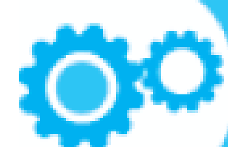


小程序文件API·电子书橱

本章主要介绍小程序文件 API 的用法，包括文件的保存、信息的获取、本地文件列表的获取、本地文件信息的获取、删除本地文件和打开指定文档。

本章学习目标

- 掌握保存临时文件的方法；
- 掌握获取文件信息的方法；
- 掌握获取本地文件列表的方法；
- 掌握获取本地文件信息的方法；
- 掌握删除本地文件的方法；
- 掌握打开指定文档的方法。



7.1 准备工作

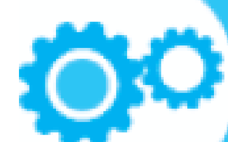




视频讲解

本项目需要将若干个 PDF 格式的电子书存放在服务器端的 books 文件夹中。由于图书版权问题，开发者可以自行准备一些电子书进行使用。

请根据“附录 A 服务器部署”中的内容进行服务器的搭建和部署工作。



7.2 项目创建





视频讲解

本项目创建选择空白文件夹 bookDemo，效果如图 7-1 所示。

单击“新建”按钮完成项目创建，然后准备手动创建页面配置文件。

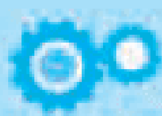
 



图 7-1　小程序项目填写效果示意图



7.3　页面配置





视频讲解

7.3.1　创建页面文件

项目创建完毕后，在根目录中会生成文件夹 pages 用于存放页面文件。一般来说首页默认命名为 index，表示小程序运行的第一个页面；其他页面名称可以自定义。本项目只需要保留首页(index)即可。

具体操作如下：

(1) 将 app.json 文件内 pages 属性中的“pages/logs/logs”删除，并删除上一行末尾的逗号。

(2) 按快捷键 Ctrl+S 保存当前修改。

7.3.2　删除和修改文件

具体操作如下：

(1) 删除 utils 文件夹及其内部所有内容。

(2) 删除 pages 文件夹下的 logs 目录及其内部所有内容。

(3) 删除 index.wxml 和 index.wxss 中的全部代码。

(4) 删除 index.js 中的全部代码，并且输入关键词 page 找到第二个选项按回车键让其自动补全函数(如图 7-2 所示)。

(5) 删除 app.wxss 中的全部代码。

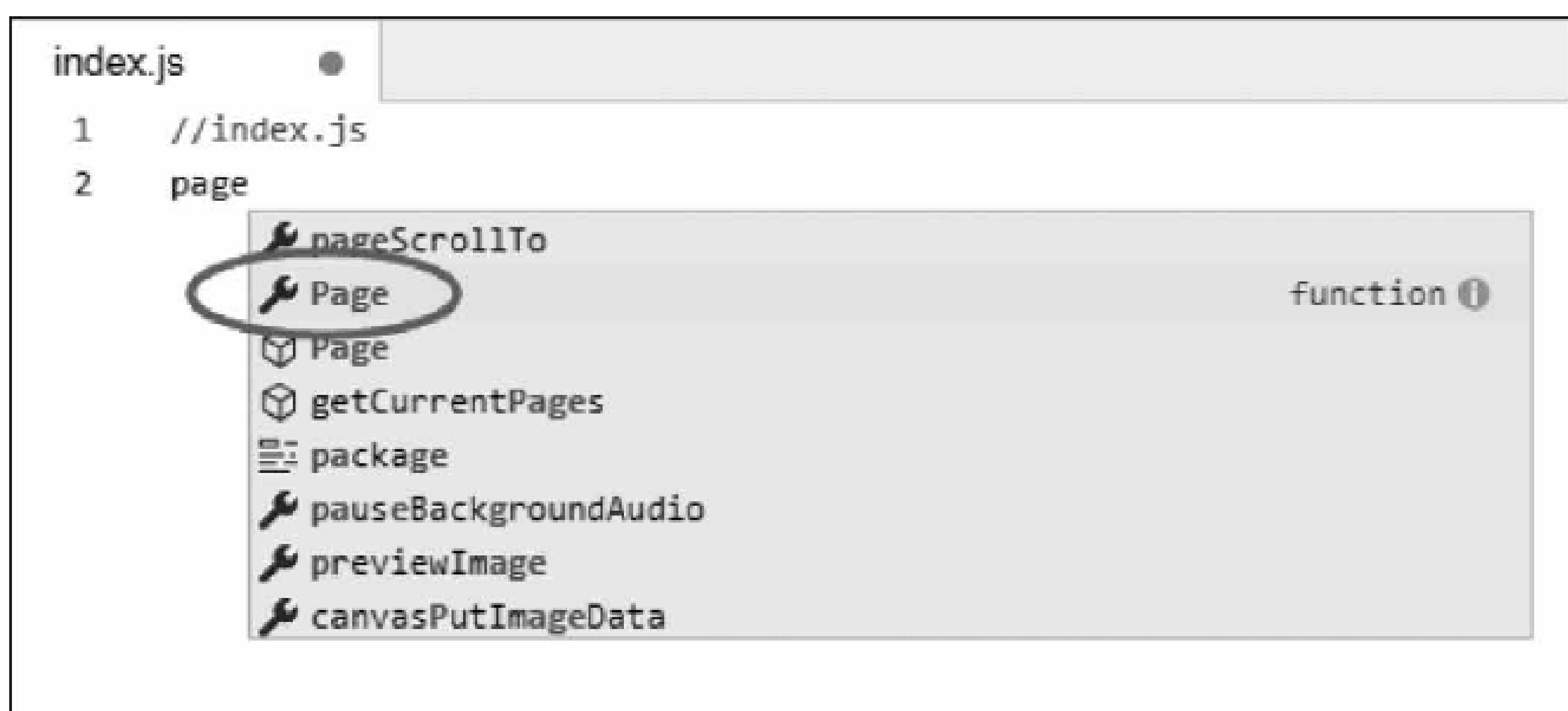


图 7-2　输入关键词创建 Page 函数

(6) 删除 app.js 中的全部代码，并且输入关键词 app 找到第一个选项按回车键让其自动补全函数(如图 7-3 所示)。

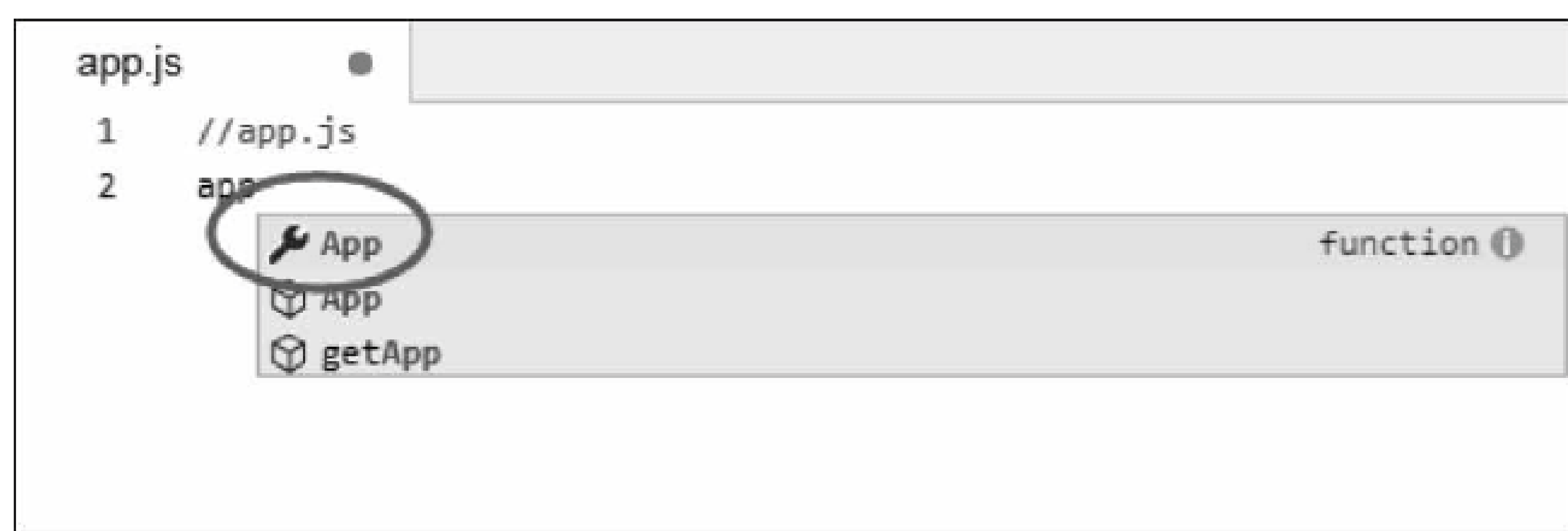
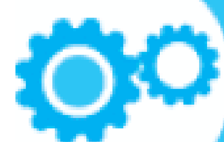


图 7-3　输入关键词创建 App 函数

本项目的文件配置就全部完成，7.4 节将正式进行页面布局和样式设计。



7.4　视图设计



7.4.1　导航栏设计



视频讲解

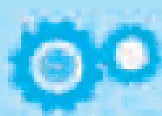
小程序默认导航栏是黑底白字的效果，因此需要在 app.json 中自定义导航栏标题和背景颜色。更改后的 app.json 文件代码如下：

```
1.  {
2.    "pages": [
3.      "pages/index/index"
4.    ],
5.    "window": {
6.      "navigationBarBackgroundColor": "#663366",
7.      "navigationBarTitleText": "我的书橱"
8.    }
9.  }
```

上述代码可以更改所有页面的导航栏标题文本为“我的书橱”、背景颜色为紫色(#663366)。预览效果如图 7-4 所示。



图 7-4　自定义导航栏效果



7.4.2 页面设计


视频讲解

页面上主要以九宫格的形式展示图书封面和标题，设计图如图 7-5 所示。

计划使用如下组件。

- 页面整体：< view >组件，并定义 class='container'；
- 图书单元区域：多个< view >组件，并定义 class='box'；
- 图书单元区域内部：一个< image >组件用于显示图书封面，一个< text >组件用于显示图书标题，垂直居中排列。

用户第一次点击图书时需要从服务器下载对应的文件，为了达到更好的用户体验，此时将隐藏图书展示容器内容并显示下载蒙层，设计图如图 7-6 所示。

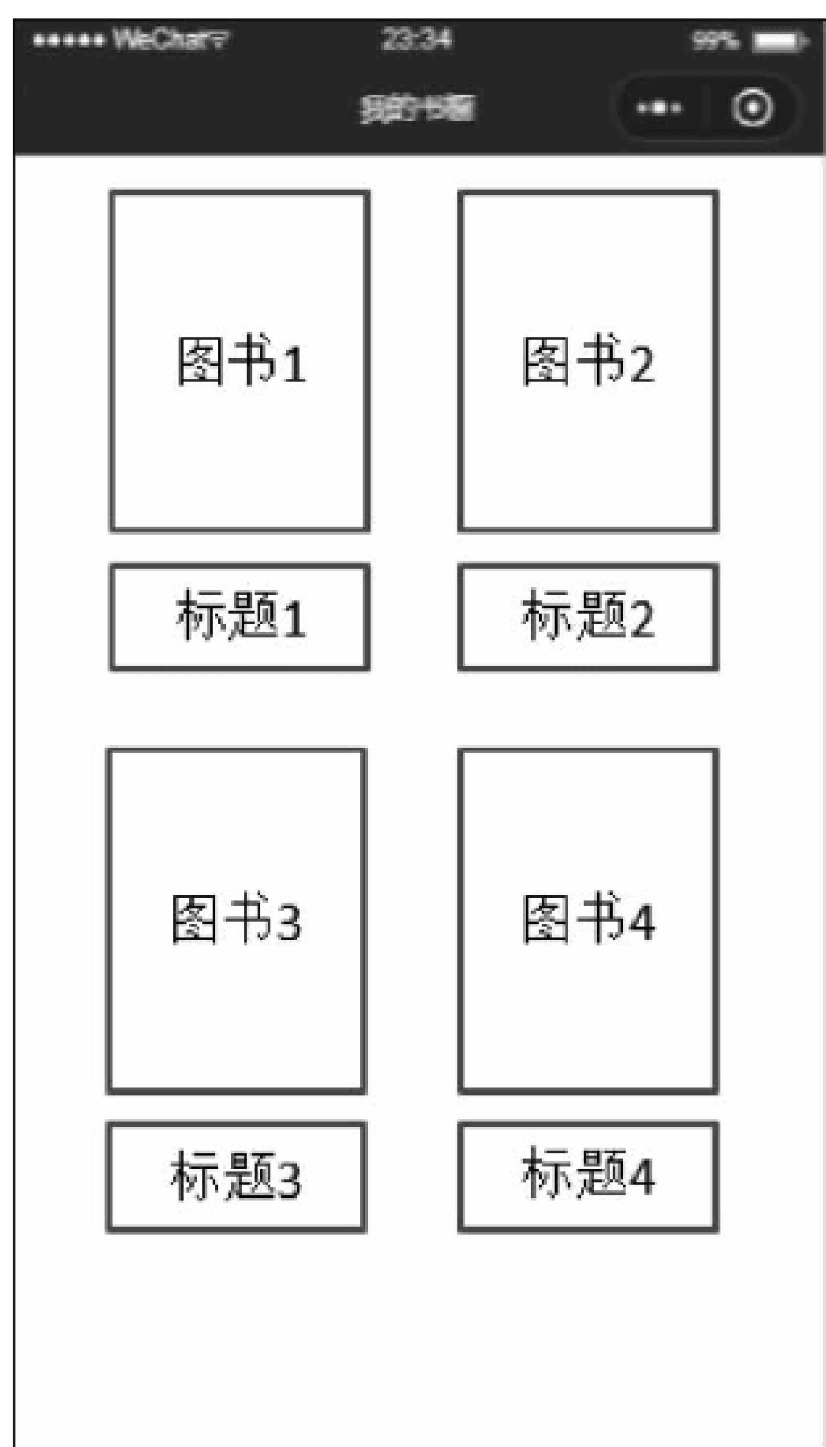


图 7-5 页面设计图 1

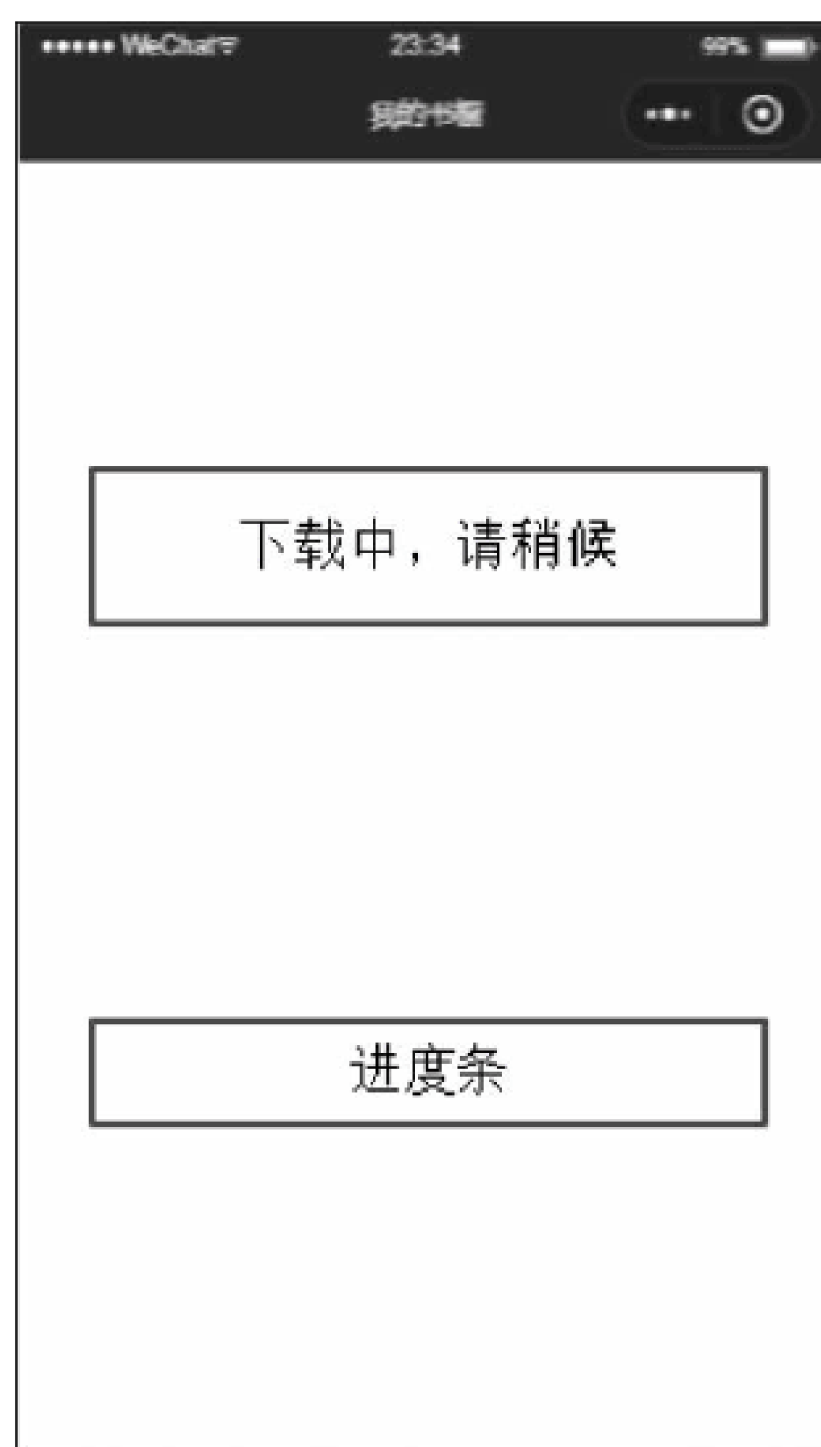


图 7-6 页面设计图 2

计划使用如下组件。

- 页面整体：< view >组件，并定义 class='loading-container'；
- 标题：< text >组件；
- 进度条：< progress >组件。

1 图书展示容器设计

首先定义页面容器(< view >)，WXML(pages/index/index.wxml)代码片段如下：

```
1.  < view class = 'book - container'>
2.  </view >
```

WXSS(pages/index/index.wxss)代码片段如下：

```
1.  / * 图书展示容器样式 * /
2.  . book - container{
3.    display: flex;                 / * flex 模型布局 * /
4.    flex - direction: row;         / * 水平排列 * /
5.    flex - wrap: wrap;             / * 允许换行 * /
6.  }
```

当前效果如图 7-7 所示。

由于还没有添加组件元素，所以尚看不出来 flex 布局模型效果。

2 图书单元区域设计

图书单元区域需要使用< view >组件实现一个可扩展的近似九宫格区域，每格中包含一个图书封面和一个图书标题文本。当前先设计第一格单元区域效果，后续使用 wx:for 属性循环添加全部内容。

WXML(pages/index/index.wxml)代码片段修改如下：

```
1.  <view class = 'book - container'>
2.    <view class = 'box'>
3.      <image src = 'https://img3.doubanio.com/view/subject/l/public/s27243455.jpg'></image>
4.      <text>Java 编程思想</text>
5.    </view>
6.  </view>
```

WXSS(pages/index/index.wxss)代码片段如下：

```
1.  /* 图书单元区域样式 */
2.  .box{
3.    width: 50%;                          /* 宽度 */
4.    height: 400rpx;                      /* 高度 */
5.    display: flex;                       /* flex 模型布局 */
6.    flex - direction: column;            /* 垂直排列 */
7.    align - items: center;               /* 水平方向居中 */
8.    justify - content: space - around;   /* 分散布局 */
9.  }
10. /* 图书封面图片样式 */
11. image{
12.   width: 200rpx;                       /* 宽度 */
13.   height: 300rpx;                      /* 高度 */
14. }
15. /* 图书标题文本样式 */
16. text{
17.   text - align: center;                /* 文本居中显示 */
18. }
```

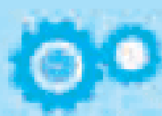
当前效果如图 7-8 所示。

3 图书下载蒙层设计

由于图书下载时的蒙层与图书展示页面不能同时出现，所以使用 wx:if 属性进行切分。

WXML(pages/index/index.wxml)代码片段修改如下：

```
1.  <!-- 下载时的蒙层 -->
2.  <view class = 'loading - container' wx:if = '{{isDownloading}}'>
3.    <text>下载中，请稍候</text>
4.     <progress percent = "{{percentNum}}" stroke - width = "6" activeColor = "#663366"
backgroundColor = "#FFFFFF" show - info active active - mode = "forwards"></progress>
5.  </view>
6.
7.  <!-- 图书展示容器 -->
8.  <view class = 'book - container' wx:else>
9.        <!-- 内容略 -->
10. </view>
```



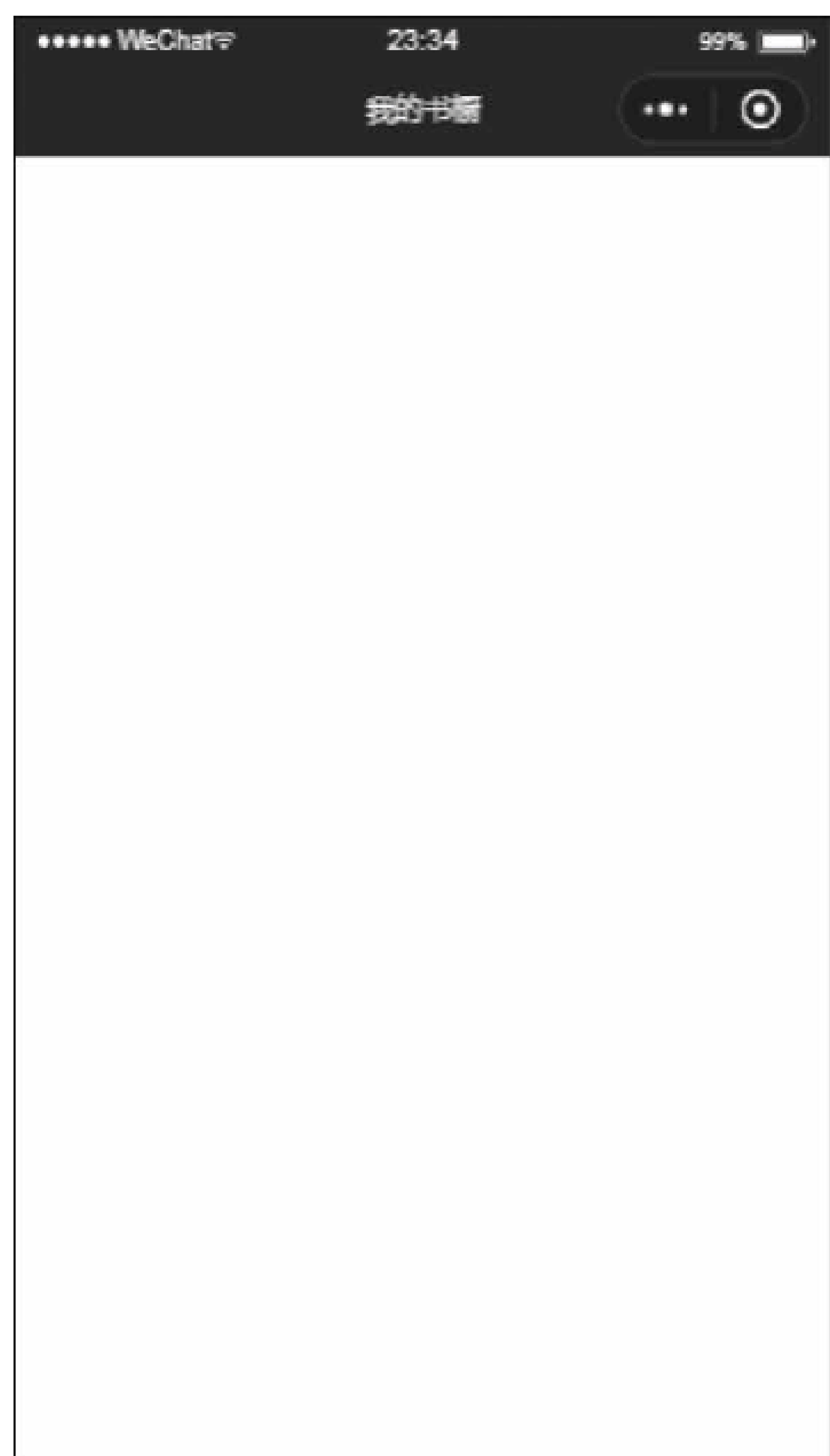


图 7-7　页面预览效果

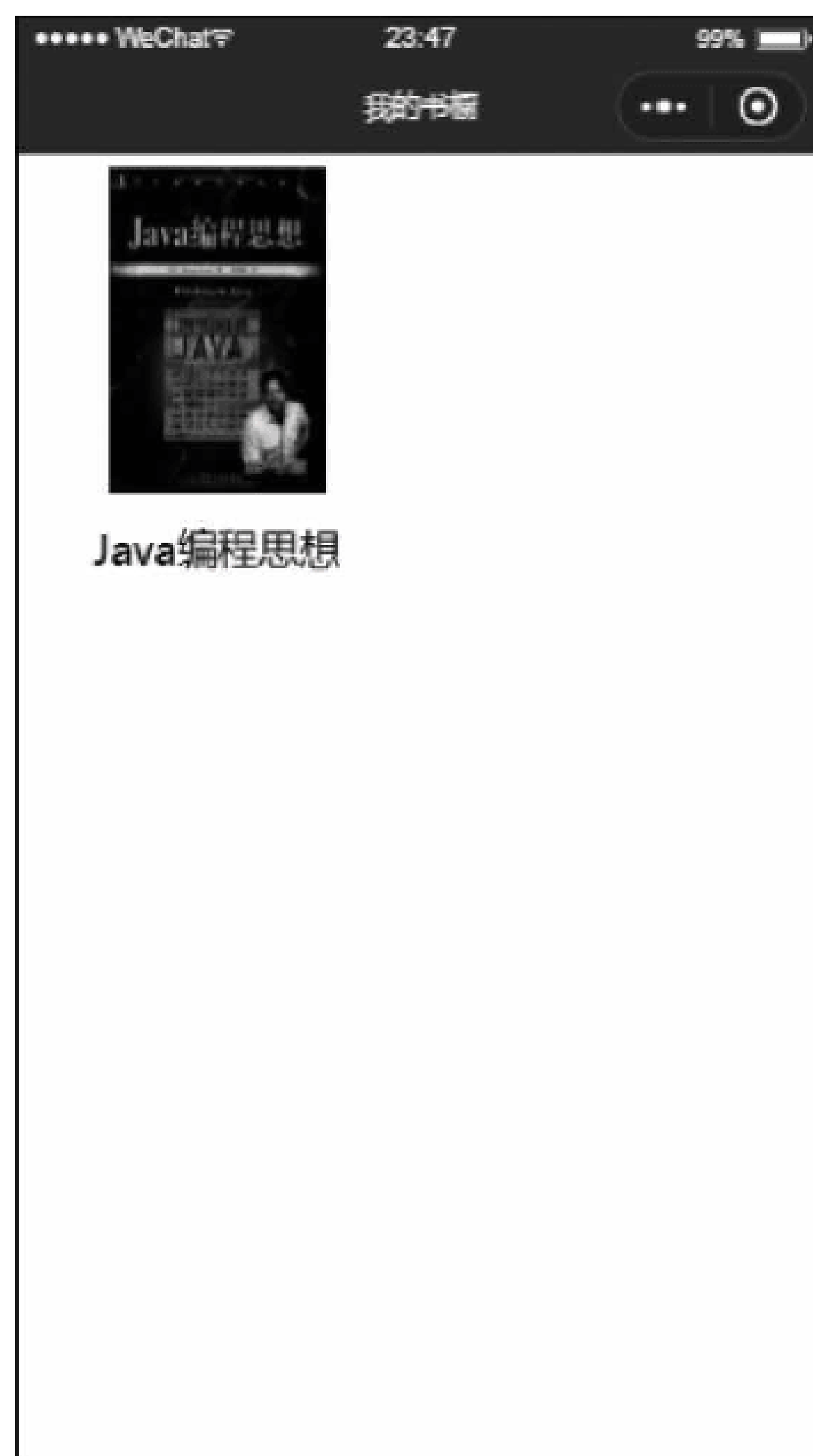


图 7-8　图书单元区域预览效果

WXSS(pages/index/index.wxss)代码片段如下：

```
1.  /* 下载时的蒙层容器样式 */
2.  .loading-container {
3.    height: 100vh;                          /* 高度 */
4.    background-color: silver;               /* 背景颜色 */
5.    display: flex;                          /* flex 模型布局 */
6.    flex-direction: column;                 /* 水平排列 */
7.    align-items: center;                    /* 水平方向居中 */
8.    justify-content: space-around;          /* 分散布局 */
9.  }
10.
11. /* 进度条样式 */
12. progress{
13.   width: 80%;                             /* 宽度 */
14. }
```

JS(pages/index/index.js)代码片段修改如下：

```
1.  Page({
2.    /**
3.     * 页面的初始数据
4.     */
5.    data: {
6.      isDownloading: false,
7.      percentNum: 0
8.    }
9.  })
```

开发者可以临时设置 isDownLoading 的值为 true，并且为 percentNum 设置一个 0～100 的数值，这样就可以提前预览进度条动画效果。当前效果如图 7-9 所示。

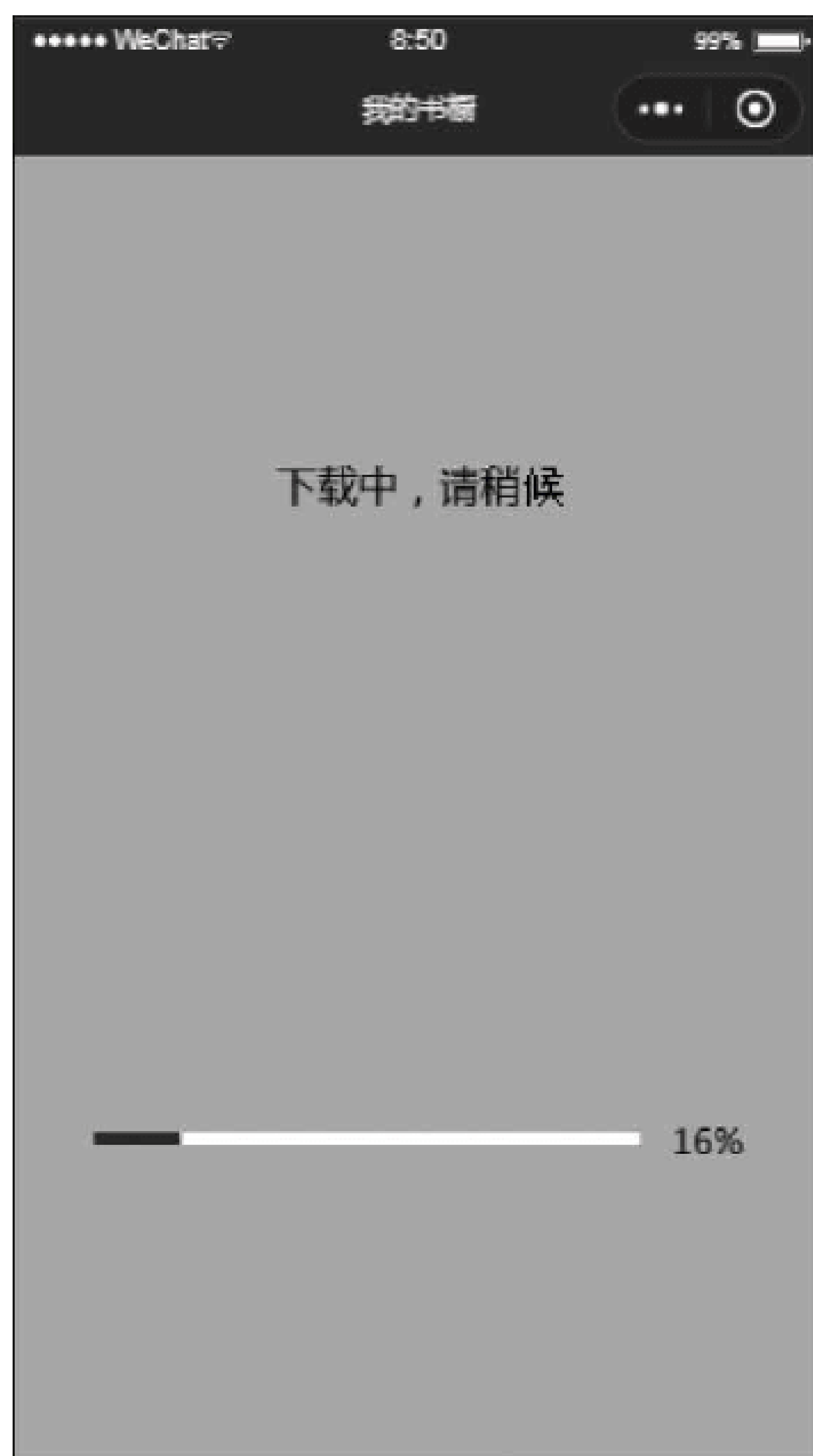


图 7-9 下载蒙层页面预览效果

测试完毕后，记得还原 isDownloading 和 percentNum 的值为初始值。

此时页面设计就全部完成了，接下来需要进行逻辑实现。



7.5 逻辑实现



7.5.1 更新图书列表



视频讲解

在图书单元区域对< view class='box'>组件添加 wx:for 属性，改写为循环展示列表。

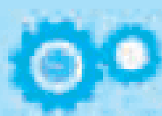
WXML(pages/index/index.wxml)代码片段修改如下：

```
1.  <view class = 'book - container' wx:else>
2.    <view class = 'box' wx:for = '{{bookList}}' wx:key = 'book{{index}}'>
3.      <image src = '{{item.poster}}'></image>
4.      <text>{{item.title}}</text>
5.    </view>
6.  </view>
```

然后在 JS 文件的 data 属性中追加 bookList 数组，用于存放图书信息。

JS(pages/index/index.js)代码片段修改如下：

```
1.  Page({
2.    /**
3.     * 页面的初始数据
4.     */
5.    data: {
6.     isDownloading: false,
7.     percentNum: 0,
8.     bookList:[
```



```
9.          {
10.           id:'001',
11.           title:'Java 编程思想',
12.           poster:'https://img3.doubanio.com/view/subject/l/public/s27243455.jpg',
13.           fileUrl:'http://localhost/books/book001.pdf'
14.         },
15.         {
16.           id: '002',
17.           title: '计算机科学概论',
18.           poster: 'https://img3.doubanio.com/view/subject/l/public/s6069865.jpg',
19.           fileUrl: 'http://localhost/books/book002.pdf'
20.         },
21.         {
22.           id: '003',
23.           title: '安徒生童话',
24.           poster: 'https://img3.doubanio.com/view/subject/l/public/s28036746.jpg',
25.           fileUrl: 'http://localhost/books/book003.pdf'
26.         },
27.         {
28.           id: '004',
29.           title: '三体(I)',
30.           poster: 'https://img1.doubanio.com/view/subject/l/public/s2768378.jpg',
31.           fileUrl: 'http://localhost/books/book004.pdf'
32.         }, {
33.           id: '005',
34.           title: '三体(II)',
35.           poster: 'https://img3.doubanio.com/view/subject/l/public/s3078482.jpg',
36.           fileUrl: 'http://localhost/books/book005.pdf'
37.         },
38.         {
39.           id: '006',
40.           title: '三体(III)',
41.           poster: 'https://img3.doubanio.com/view/subject/l/public/s26012674.jpg',
42.           fileUrl: 'http://localhost/books/book006.pdf'
43.         }
44.       ]
45.     },
46. })
```

其中封面图片素材来自豆瓣读书频道(https://book.douban.com/),文件下载为本地模拟服务器效果(http://localhost/books/bookXXX.pdf),开发者可以自行更改存放图书的路径地址和电子书名称。

运行效果如图 7-10 所示。

由图可见,当前已经可以展示全部图书列表。



图 7-10 更新图书列表效果

7.5.2 封装提示消息



视频讲解

由于多处会使用提示消息模态框,所以在 JS 文件中对其进行封装,以方便后续使用。

在 JS 文件中添加自定义函数 showTips,JS(pages/index/index.js)代码片段如下:

```
1.  Page({
2.    /**
3.     * 封装 showModal 方法
4.     */
5.    showTips: function(content) {
6.      wx.showModal({
7.        title: '提醒',
8.        content: content,
9.        showCancel: false
10.     })
11.   },
12. })
```

这样在其他函数中只需要调用 this.showTips('自定义的提示信息')就可以方便地实现模态框提示效果了。

7.5.3 打开指定图书



视频讲解

当用户点击图书时，如果已经下载过，可以直接打开，否则需要进行下载、保存，然后才能打开。因此在这两种不同的情况判断中，最后都需要实现打开指定图书的功能。为了避免代码重复，这里对该功能进行封装。

在 JS 文件中添加自定义函数 openBook，JS(pages/index/index.js)代码片段如下：

```
1.  Page({
2.    /**
3.     * 打开图书
4.     */
5.    openBook: function(path) {
6.      wx.openDocument({
7.        filePath: path,
8.        success: function(res) {
9.          console.log('打开图书成功')
10.       },
11.       fail: function(error) {
12.         console.log(error);
13.       }
14.     })
15.   },
16. })
```

这样在其他函数中只需要调用 this.openBook('指定的图书地址')就可以方便地实现打开图书的效果了。

7.5.4 保存下载的图书

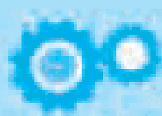


视频讲解

这里对保存图书功能进行封装，当图书下载完毕后根据临时地址进行文件的保存。

在 JS 文件中添加自定义函数 saveBook，JS(pages/index/index.js)代码片段如下：

```
1.  Page({
2.  /**
3.     * 保存图书
```

```
4.     */
5.     saveBook: function(id, path) {
6.       var that = this
7.       wx.saveFile({
8.         tempFilePath: path,
9.         success: function(res) {
10.          //将文件地址保存到本地缓存中,下次直接打开
11.          let newPath = res.savedFilePath
12.          wx.setStorageSync(id, newPath)
13.          //打开图书
14.          that.openBook(newPath)
15.        },
16.        fail: function(error) {
17.          console.log(error)
18.          that.showTips('文件保存失败!')
19.        }
20.      })
21.    },
22. })
```

这样在其他函数中只需要调用 this.saveBook('下载后的临时地址')就可以方便地实现保存并打开图书的效果了。需要注意的是,应尽量避免在 wx.开头的 API 内部使用 this 关键词,因此需要声明 var that = this 来进行方法的调用。

7.5.5 下载并阅读图书



视频讲解

首先修改图书单元区域,为其添加 bindtap 事件,触发自定义函数 readBook。WXML(pages/index/index.wxml)代码片段修改如下:

```
1.  <!-- 图书展示容器 -->
2.  <view class='book-container' wx:else>
3.    <!-- 图书单元区域 -->
4.    <view class='box' wx:for='{{bookList}}' wx:key='book{{index}}' data-file='{{item.
fileUrl}}' data-id='{{item.id}}' bindtap='readBook'>
5.      <!-- 内容略 -->
6.    </view>
7.  </view>
```

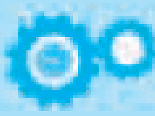
然后在 JS 文件中添加自定义函数 readBook,JS(pages/index/index.js)代码片段如下:

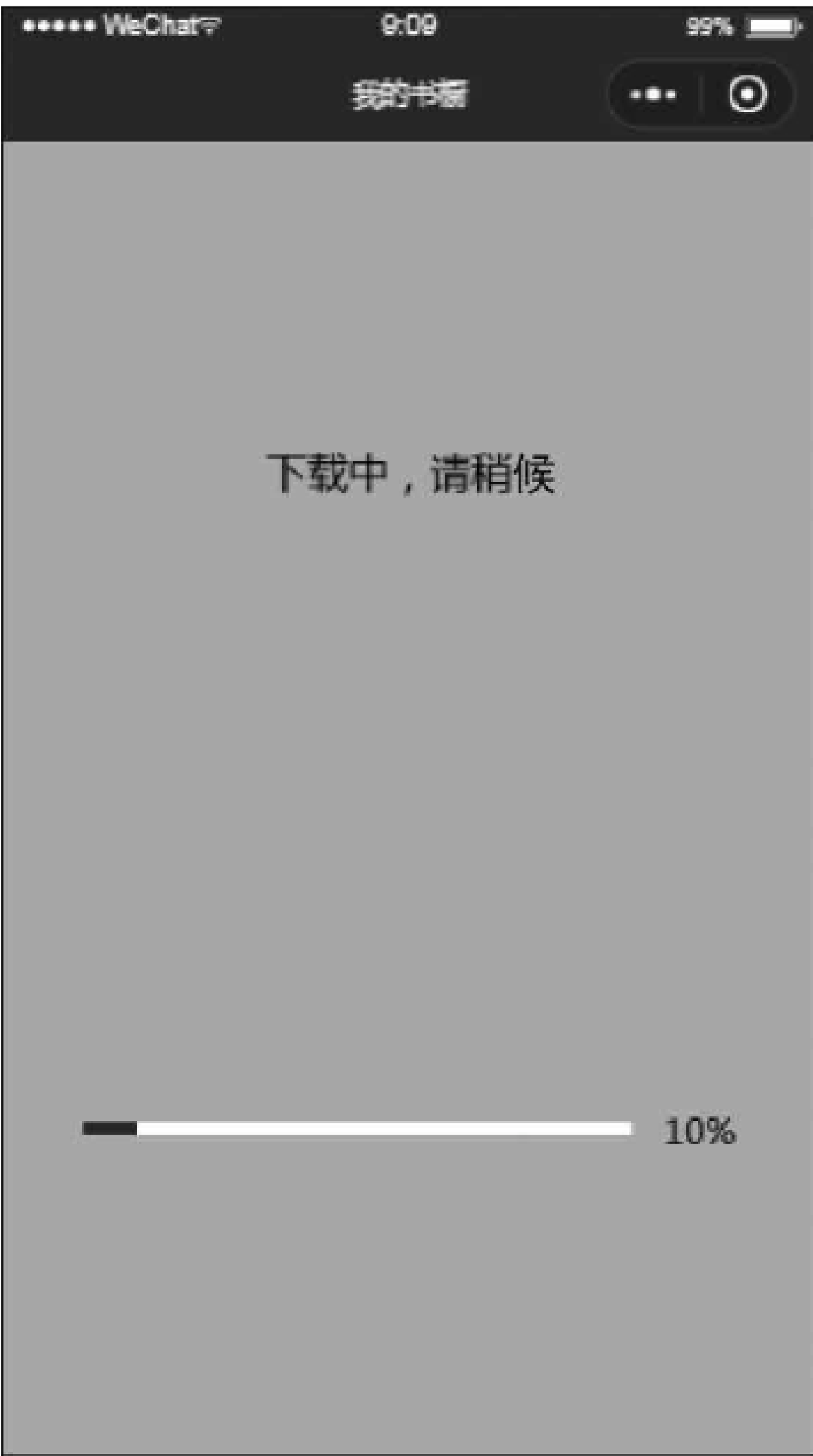
```
1.  Page({
2.    /**
3.     * 阅读图书
4.     */
5.    readBook: function(e) {
6.      var that = this
7.      //获取当前点击图书的 ID
8.      let id = e.currentTarget.dataset.id
9.      //获取当前点击图书的 URL 地址
10.     let fileUrl = e.currentTarget.dataset.file
11.     //查看本地缓存
12.     let path = wx.getStorageSync(id)
13.     //未曾下载过
14.     if (path == '') {
15.       //切换到下载时的蒙层
```

```
16.         that.setData({
17.           isDownloading: true
18.         })
19.         //先下载图书
20.         const downloadTask = wx.downloadFile({
21.           url: fileUrl,
22.           success: function(res) {
23.             //关闭下载时的蒙层
24.             that.setData({
25.               isDownloading: false
26.             })
27.             //下载成功
28.             if (res.statusCode == 200) {
29.               //获取地址
30.               path = res.tempFilePath
31.               //保存并打开图书
32.               that.saveBook(id, path)
33.             }
34.             //连上了服务器,下载失败
35.             else {
36.               that.showTips('暂时无法下载!')
37.             }
38.           },
39.           //请求失败
40.           fail: function(e) {
41.             //关闭下载时的蒙层
42.             that.setData({
43.               isDownloading: false
44.             })
45.             that.showTips('无法连接到服务器!')
46.           }
47.         })
48.         //监听当前文件的下载进度
49.         downloadTask.onProgressUpdate(function(res) {
50.           let progress = res.progress;
51.           that.setData({
52.             percentNum: progress
53.           })
54.         })
55.       }
56.       //之前下载过的,直接打开
57.       else {
58.         //打开图书
59.         that.openBook(path)
60.       }
61.     },
62. })
```

运行效果如图 7-11 所示。

其中图 7-11(a)是首次点击下载图书的页面,进度条会不断更新;图 7-11(b)是下载地址有误时的提示信息,此时无法下载资源文件;图 7-11(c)是已经下载成功但文件保存失败时的提示信息,需要注意本地文件最大限制为 10MB;图 7-11(d)是服务器无法响应时的提示信息,需要检查服务器地址是否正确,以及服务器是否开启;图 7-11(e)是成功下载、保存并打开电子图书的页面。





(a) 正在下载图书



(b) 特殊情况：下载地址有误



(c) 特殊情况：文件过大保存失败



(d) 特殊情况：服务器无响应

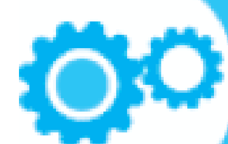
图 7-11　点击图书效果



(e) 下载成功并打开

图 7-11　（续）

需要注意的是，localhost 并非真正的服务器地址，因此仅可在 web 开发者工具中模拟使用。开发者可以后期将文件上传到服务器上，修改文件下载 URL 地址，再进行真机运行。



7.6　完整代码展示

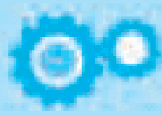


app.json 文件的完整代码如下：

```
1.  {
2.    "pages": [
3.      "pages/index/index"
4.    ],
5.    "window": {
6.      "navigationBarBackgroundColor": "#663366",
7.      "navigationBarTitleText": "我的书橱"
8.    }
9.  }
```

WXML 文件(pages/index/index.wxml)的完整代码如下：

```
1.  <!-- 下载时的蒙层 -->
2.  <view class='loading-container' wx:if='{{isDownloading}}'>
3.    <text>下载中,请稍候</text>
4.     <progress percent="{{percentNum}}" stroke-width="6" activeColor="#663366"
backgroundColor="#FFFFFF" show-info active active-mode="forwards"></progress>
5.  </view>
6.
7.  <!-- 图书展示容器 -->
8.  <view class='book-container' wx:else>
9.    <!-- 图书单元区域 -->
10.   <view class='box' wx:for='{{bookList}}' wx:key='book{{index}}' data-file='{{item.
```



```
fileUrl}}' data - id = '{{item.id}}' bindtap = 'readBook'>
11.       <!-- 图书封面 -->
12.       < image src = '{{item.poster}}'></image >
13.       <!-- 图书标题文本 -->
14.       < text >{{item.title}}</text >
15.     </view >
16.   </view >
```

WXSS 文件(pages/index/index.wxss)的完整代码如下：

```
1.  /* 下载时的蒙层容器样式 */
2.  .loading - container {
3.    height: 100vh;                          /* 高度 */
4.    background - color: silver;             /* 背景颜色 */
5.    display: flex;                          /* flex 模型布局 */
6.    flex - direction: column;               /* 水平排列 */
7.    align - items: center;                  /* 水平方向居中 */
8.    justify - content: space - around;      /* 分散布局 */
9.  }
10.
11. /* 进度条样式 */
12. progress{
13.   width: 80 % ;                           /* 宽度 */
14. }
15.
16. /* 图书展示容器样式 */
17. .book - container {
18.   display: flex;                          /* flex 模型布局 */
19.   flex - direction: row;                  /* 水平排列 */
20.   flex - wrap: wrap;                      /* 允许换行 */
21. }
22.
23. /* 图书单元区域样式 */
24. .box {
25.   width: 50 % ;                           /* 宽度 */
26.   height: 400rpx;                         /* 高度 */
27.   display: flex;                          /* flex 模型布局 */
28.   flex - direction: column;               /* 垂直排列 */
29.   align - items: center;                  /* 水平方向居中 */
30.   justify - content: space - around;      /* 分散布局 */
31. }
32.
33. /* 图书封面图片样式 */
34. image {
35.   width: 200rpx;                          /* 宽度 */
36.   height: 300rpx;                         /* 高度 */
37. }
38.
39. /* 图书标题文本样式 */
40. text {
41.   text - align: center;                   /* 文本居中显示 */
42. }
```

JS 文件(pages/index/index.js)的完整代码如下：

```
1.  Page({
```

```
2.     /**
3.      * 页面的初始数据
4.      */
5.     data: {
6.       isDownloading: false,
7.       percentNum: 0,
8.       bookList: [{
9.           id: '001',
10.          title: 'Java 编程思想',
11.          poster: 'https://img3.doubanio.com/view/subject/l/public/s27243455.jpg',
12.          fileUrl: 'http://localhost/books/book001.pdf'
13.        },
14.        {
15.          id: '002',
16.          title: '计算机科学概论',
17.          poster: 'https://img3.doubanio.com/view/subject/l/public/s6069865.jpg',
18.          fileUrl: 'http://localhost/books/book002.pdf'
19.        },
20.        {
21.          id: '003',
22.          title: '安徒生童话',
23.          poster: 'https://img3.doubanio.com/view/subject/l/public/s28036746.jpg',
24.          fileUrl: 'http://localhost/books/book003.pdf'
25.        },
26.        {
27.          id: '004',
28.          title: '三体(I)',
29.          poster: 'https://img1.doubanio.com/view/subject/l/public/s2768378.jpg',
30.          fileUrl: 'http://localhost/books/book004.pdf'
31.        }, {
32.          id: '005',
33.          title: '三体(II)',
34.          poster: 'https://img3.doubanio.com/view/subject/l/public/s3078482.jpg',
35.          fileUrl: 'http://localhost/books/book005.pdf'
36.        },
37.        {
38.          id: '006',
39.          title: '三体(III)',
40.          poster: 'https://img3.doubanio.com/view/subject/l/public/s26012674.jpg',
41.          fileUrl: 'http://localhost/books/book006.pdf'
42.        }
43.      ]
44.    },
45.    /**
46.     * 打开图书
47.     */
48.    openBook: function(path) {
49.      wx.openDocument({
50.        filePath: path,
51.        success: function(res) {
52.          console.log('打开图书成功')
53.        },
54.        fail: function(error) {
55.          console.log(error);
56.        }
```

```
57.     })
58.   },
59.   /**
60.    * 保存图书
61.    */
62.   saveBook: function(id, path) {
63.     var that = this
64.     wx.saveFile({
65.       tempFilePath: path,
66.       success: function(res) {
67.         //将文件地址保存到本地缓存中,下次直接打开
68.         let newPath = res.savedFilePath
69.         wx.setStorageSync(id, newPath)
70.         //打开图书
71.         that.openBook(newPath)
72.       },
73.       fail: function(error) {
74.         console.log(error)
75.         that.showTips('文件保存失败!')
76.       }
77.     })
78.   },
79.   /**
80.    * 阅读图书
81.    */
82.   readBook: function(e) {
83.     var that = this
84.     //获取当前点击图书的 ID
85.     let id = e.currentTarget.dataset.id
86.     //获取当前点击图书的 URL 地址
87.     let fileUrl = e.currentTarget.dataset.file
88.     //查看本地缓存
89.     let path = wx.getStorageSync(id)
90.     //未曾下载过
91.     if (path == '') {
92.       //切换到下载时的蒙层
93.       that.setData({
94.         isDownloading: true
95.       })
96.       //先下载图书
97.       const downloadTask = wx.downloadFile({
98.         url: fileUrl,
99.         success: function(res) {
100.          //关闭下载时的蒙层
101.          that.setData({
102.            isDownloading: false
103.          })
104.          //下载成功
105.          if (res.statusCode == 200) {
106.            //获取地址
107.            path = res.tempFilePath
108.            //保存并打开图书
109.            that.saveBook(id, path)
110.          }
111.          //连上了服务器,下载失败
```

```
112.           else {
113.             that.showTips('暂时无法下载!')
114.           }
115.         },
116.         //请求失败
117.         fail: function(e) {
118.           //关闭下载时的蒙层
119.           that.setData({
120.             isDownloading: false
121.           })
122.           that.showTips('无法连接到服务器!')
123.         }
124.       })
125.       //监听当前文件的下载进度
126.       downloadTask.onProgressUpdate(function(res) {
127.         let progress = res.progress;
128.         that.setData({
129.           percentNum: progress
130.         })
131.       })
132.     }
133.     //之前下载过的,直接打开
134.     else {
135.       //打开图书
136.       that.openBook(path)
137.     }
138.   },
139.
140.   /**
141.    * 封装 showModal 方法
142.    */
143.   showTips: function(content) {
144.     wx.showModal({
145.       title: '提醒',
146.       content: content,
147.       showCancel: false
148.     })
149.   },
150. })
```

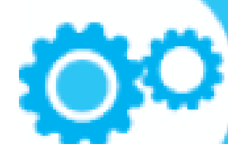
第8章  Chapter 8

小程序数据API · 医疗急救卡

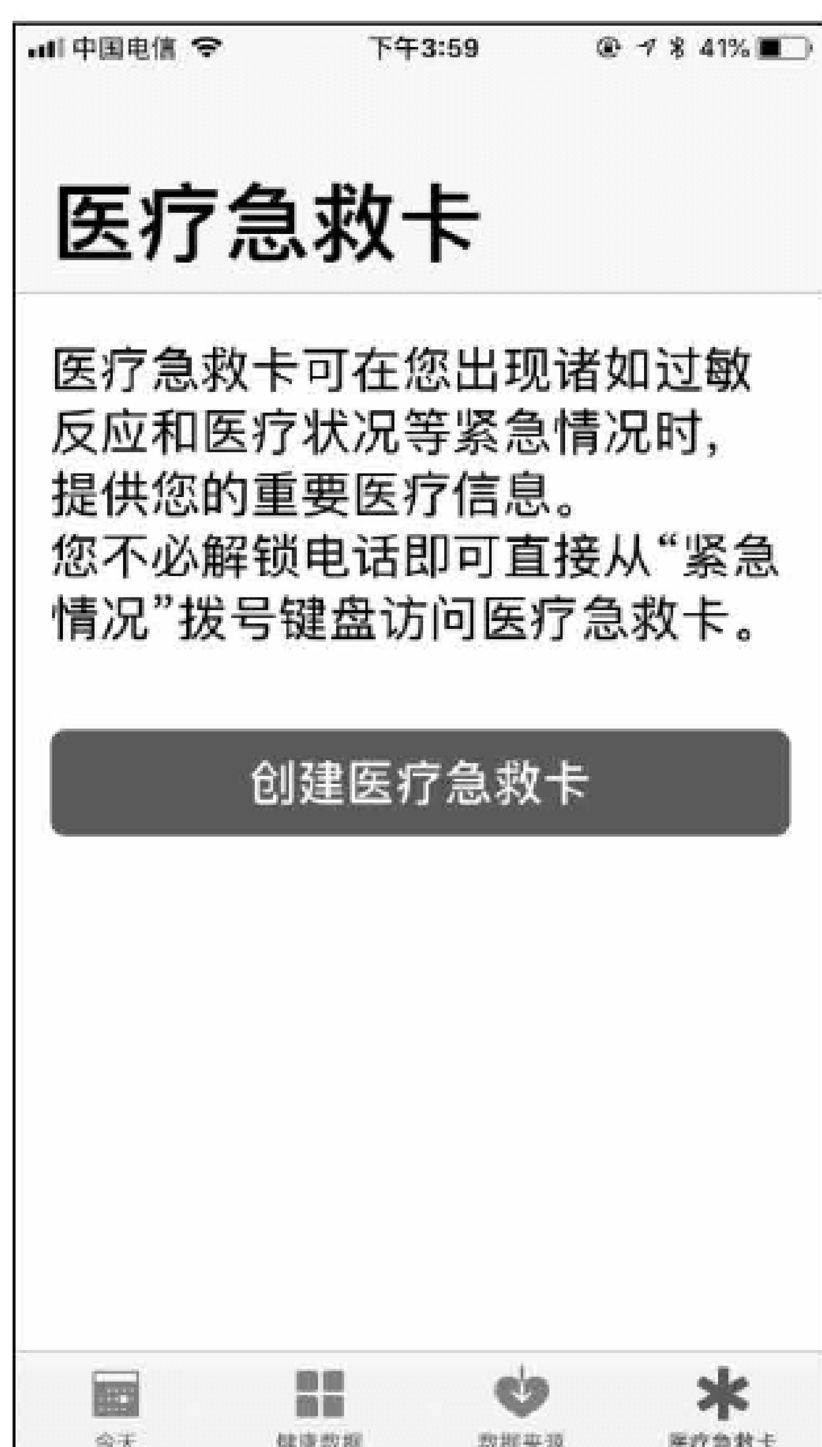
本章将介绍使用小程序表单组件仿 iOS 健康 App 创建一个简易版的医疗急救卡小程序。

本章学习目标

- 学习小程序表单系列组件的用法；
- 学习本地数据存储和读取的方法；
- 学习拨打指定电话号码的实现。

 8.1 项目介绍 

iOS 系统自带的健康 App 中包含了“医疗急救卡”页面，旨在用户出现过敏或突发医疗情况时为急救人员提供重要的医疗健康信息和紧急联系人，如图 8-1 所示。



(a) 尚未创建急救卡效果



(b) 创建急救卡过程



(c) 急救卡创建完毕

图 8-1　iOS 健康 App“医疗急救卡”页面真机截屏

 8.2 项目创建 


视频讲解

本项目创建选择空白文件夹 firstaidCard，效果如图 8-2 所示。

单击“新建”按钮完成项目创建，然后准备手动创建页面配置文件。



图 8-2 小程序项目填写效果示意图

 8.3 页面配置 

8.3.1 创建页面文件


视频讲解

项目创建完毕后，在根目录中会生成文件夹 pages 用于存放页面文件。一般来说首页默认命名为 index，表示小程序运行的第一个页面；其他页面名称可以自定义。本项目需要保留首页（index），并新增医疗急救卡创建页（form）。

具体操作如下：

（1）将 app.json 文件内 pages 属性中的“pages/logs/logs”替换成“pages/form/form”。

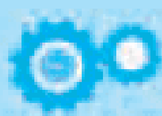
（2）按快捷键 Ctrl+S 保存当前修改，此时 form 目录会自动生成到 pages 文件夹下。

8.3.2 删除和修改文件

具体操作如下：

（1）删除 utils 文件夹及其内部所有内容。

（2）删除 pages 文件夹下的 logs 目录及其内部所有内容。



（3）删除 index. wxml 和 index. wxss 中的全部代码。

（4）删除 index. js 中的全部代码，并且输入关键词 page 找到第二个选项按回车键让其自动补全函数（如图 8-3 所示）。

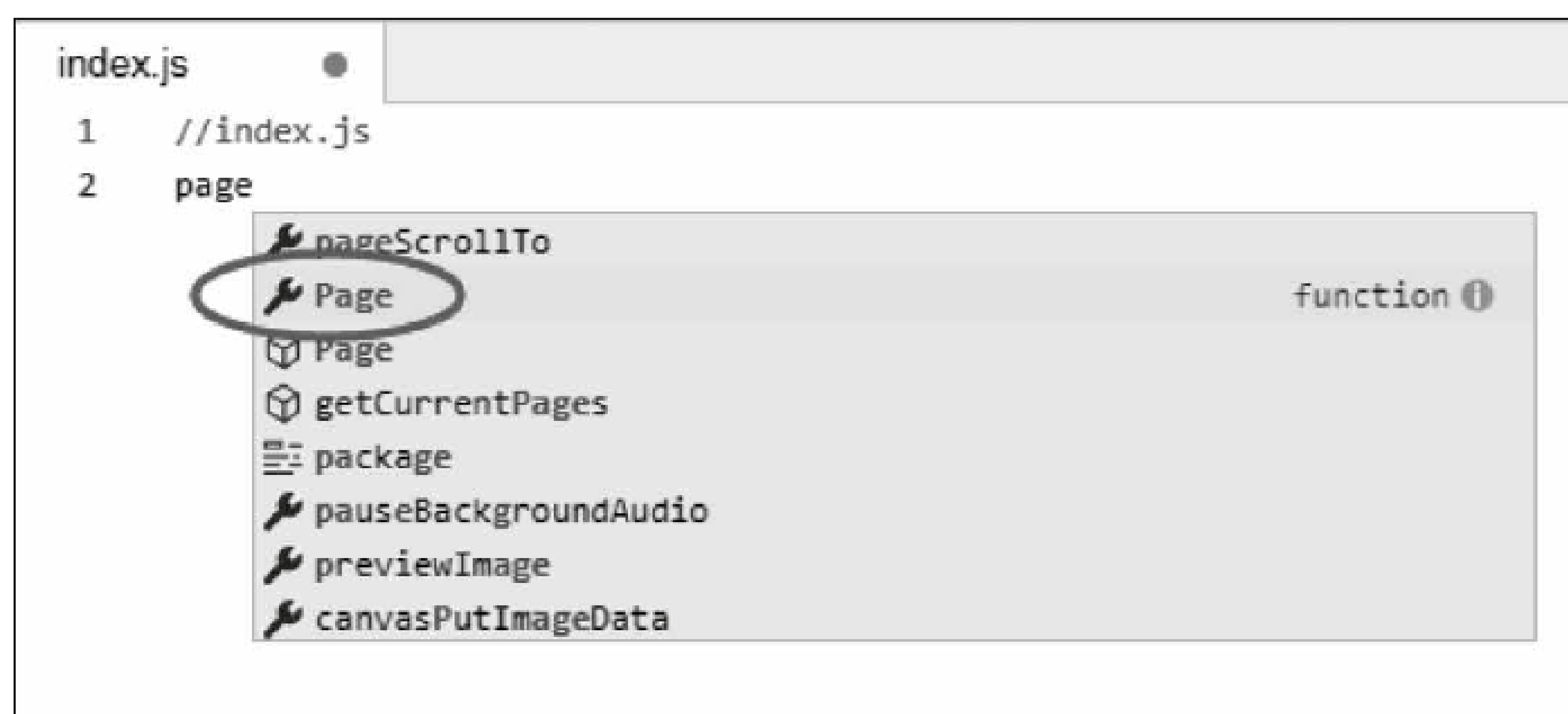


图 8-3　输入关键词创建 Page 函数

（5）删除 app. wxss 中的全部代码。

（6）删除 app. js 中的全部代码，并且输入关键词 app 找到第一个选项按回车键让其自动补全函数（如图 8-4 所示）。

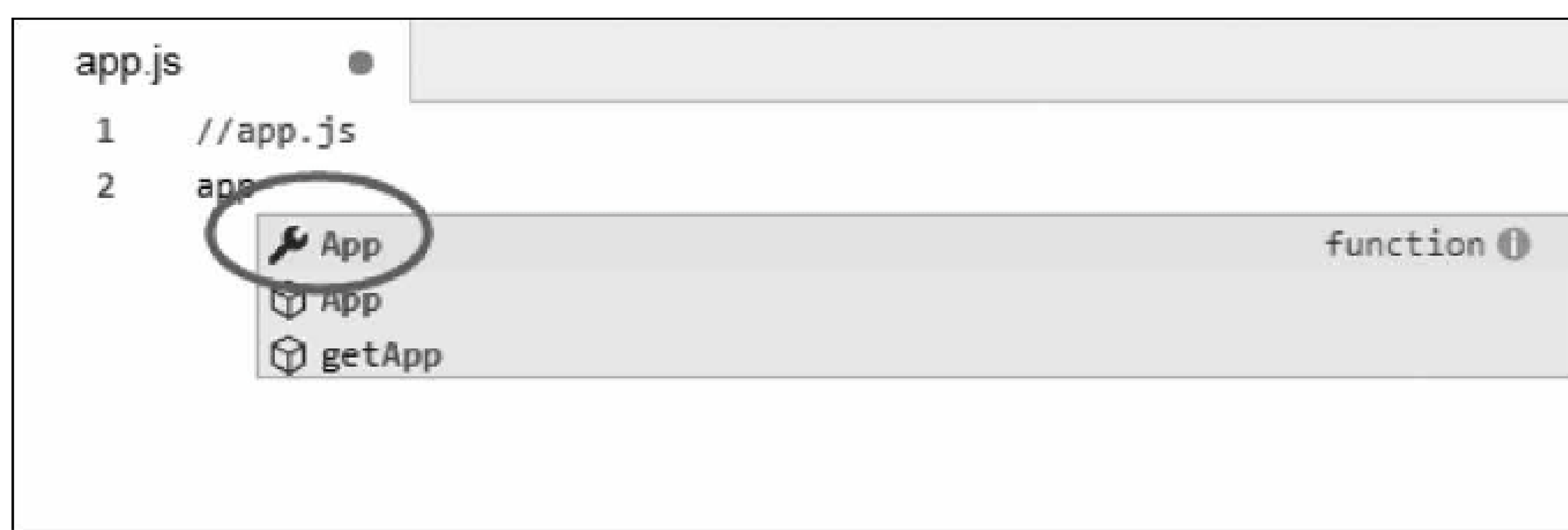


图 8-4　输入关键词创建 App 函数

完成后的目录结构如图 8-5 所示。

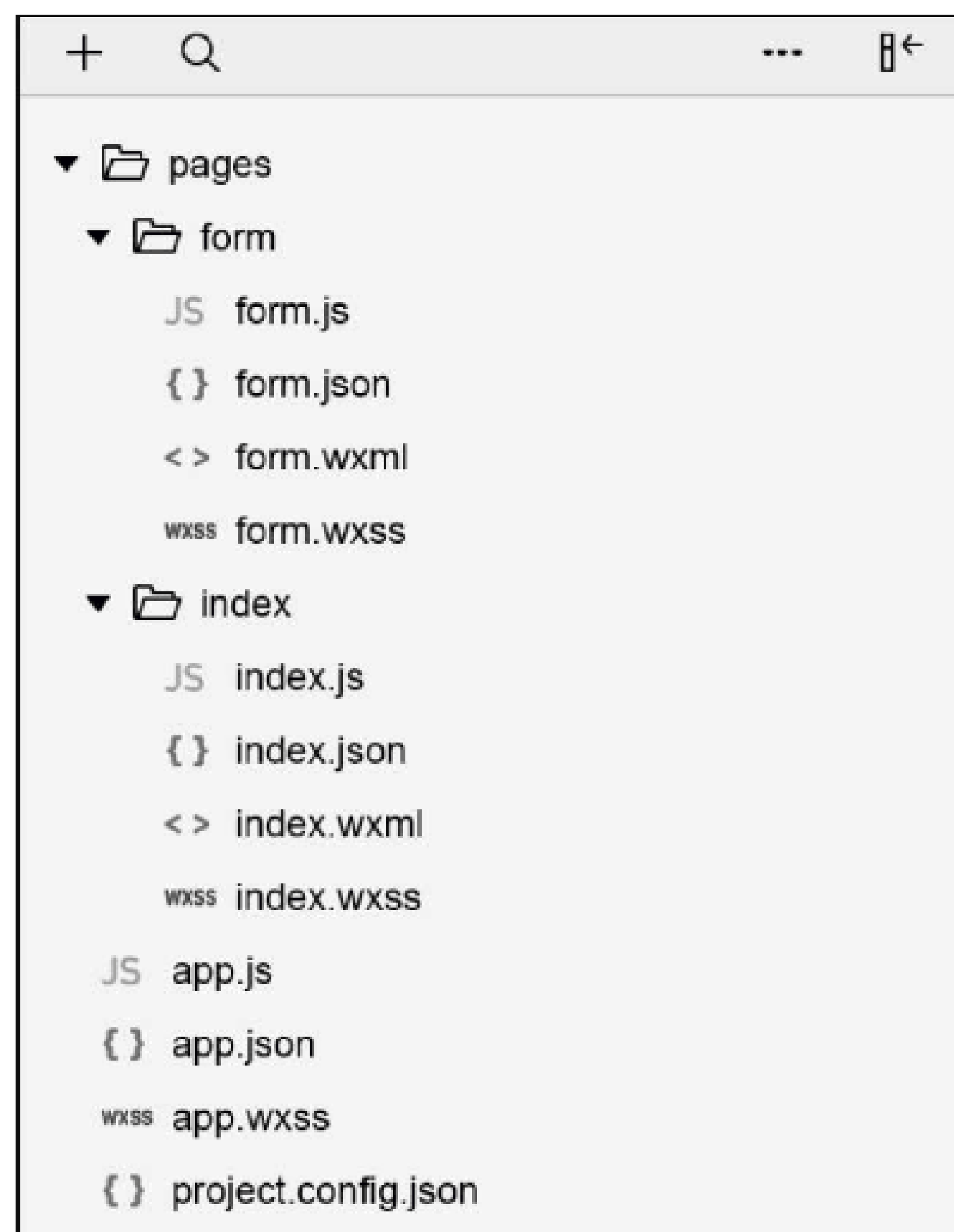
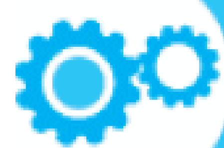


图 8-5　页面文件创建完成

此时文件配置就全部完成，8.4 节将正式进行页面布局和样式设计。



8.4 视图设计



8.4.1 导航栏设计



视频讲解

小程序默认导航栏是黑底白字的效果，因此需要在 app.json 中自定义导航栏标题和背景颜色。更改后的 app.json 文件代码如下：

```
1.  {
2.    "pages": [
3.      "pages/index/index",
4.      "pages/form/form"
5.    ],
6.    "window":{
7.      "navigationBarBackgroundColor":"#FF2D55",
8.      "navigationBarTitleText": "医疗急救卡"
9.    }
10. }
```

上述代码可更改所有页面导航栏标题文本为“医疗急救卡”、背景颜色为红色(#FF2D55)。

预览效果如图 8-6 所示。



图 8-6 自定义导航栏效果

8.4.2 页面设计



视频讲解

1 首页设计

首页分两种情况，具体解释如下。

- 尚未创建医疗急救卡：此时只显示一个“创建医疗急救卡”按钮；
- 已经创建医疗急救卡：此时显示用户的医疗信息，最下方显示“打电话给紧急联系人”和“编辑医疗急救卡”两个按钮。

页面设计如图 8-7 所示。

图 8-7(a)设计图计划使用如下组件。

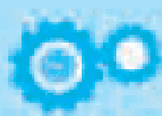
- 页面整体：< view >组件，并定义 class='container'；
- 按钮：< button >组件。

图 8-7(b)设计图计划使用如下组件。

- 每条信息：< view >组件，并定义 class='box'，根据其排列方向再定义 class 为 row 或 col；
- 医疗信息标题：< label >组件；
- 医疗信息文本：< text >组件；
- 按钮：< button >组件。

首先定义尚未创建医疗急救卡时的效果，WXML(pages/index/index.wxml)代码片段如下：

```
1.  <!-- 尚未创建医疗急救卡显示该页面 -->
2.  <view class = 'container'>
3.    <button>创建医疗急救卡</button>
4.  </view>
```



(a) 尚未创建医疗急救卡的页面效果　(b) 已经创建医疗急救卡的页面效果

图 8-7　页面设计图

WXSS(pages/index/index.wxss)代码片段如下：

```
1.  .container{
2.     height: 100vh;                 /* 高 100 视窗,写成 100% 无效 */
3.     display: flex;                 /* flex 布局模型 */
4.     flex-direction: column;        /* 垂直布局 */
5.     align-items: center;           /* 水平方向居中 */
6.     justify-content: center;       /* 垂直方向居中 */
7.  }
```

由于按钮样式需要与 form 页面统一，可以写到 app.wxss 文件中。

app.wxss 代码片段如下：

```
1.  /* 按钮样式 */
2.  button{
3.     background-color: #FF2D55;     /* 背景颜色为红色 */
4.     color: white;                  /* 字体颜色为白色 */
5.     margin: 20rpx;                 /* 外边距 */
6.  }
```

当前效果如图 8-8 所示。

由图可见，此时页面初始效果已经完成。接下来需要使用 wx:if 和 wx:else 属性切换医疗急救卡创建完成后的页面设计效果。

WXML(pages/index/index.wxml)代码片段修改如下：

```
1.  <!-- 尚未创建医疗急救卡显示该页面 -->
2.  <view class='container' wx:if='{{!myCard}}'>
3.     <button>创建医疗急救卡</button>
4.  </view>
```

```
5.  <!-- 医疗急救卡创建完成后显示该页面 -->
6.  <view wx:else>
7.    <view class = 'row box'>
8.      <label>出生日期</label>
9.      <text>2000 - 01 - 01 </text>
10.     <label>血型</label>
11.     <text>A 型</text>
12.   </view>
13.   <view class = 'row box'>
14.     <label>身高</label>
15.     <text>100 厘米</text>
16.     <label>体重</label>
17.     <text>50 千克</text>
18.   </view>
19.   <view class = 'col box'>
20.     <label>医疗状况</label>
21.     <text>无</text>
22.   </view>
23.   <view class = 'col box'>
24.     <label>医疗笔记</label>
25.     <text>无</text>
26.   </view>
27.   <view class = 'col box'>
28.     <label>过敏反应</label>
29.     <text>无</text>
30.   </view>
31.   <view class = 'col box'>
32.     <label>用药</label>
33.     <text>无</text>
34.   </view>
35.   <view class = 'row box'>
36.     <label>器官捐赠者</label>
37.     <text>否</text>
38.   </view>
39.   <view class = 'row box'>
40.     <label>紧急联系人</label>
41.     <text>10086 </text>
42.   </view>
43.   <button>打电话给紧急联系人</button>
44.   <button>编辑医疗急救卡</button>
45. </view>
```

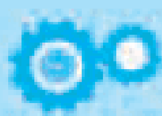


图 8-8 首页(尚未填写医疗急救卡)预览效果

为了预览测试效果,上述代码中的医疗数据均为临时填入,后续会改为动态数据。

app.wxss 增加代码片段如下:

```
1.  /* 标签样式 */
2.  label {
3.    color: #FF2D55;                 /* 字体颜色为红色 */
4.    margin-right:25rpx;             /* 右边距 */
5.  }
6.  /* 水平布局 */
7.  .row {
8.    display: flex;                  /* flex 布局模型 */
9.    flex-direction: row;            /* 水平布局 */
10. }
11. /* 垂直布局 */
12. .col {
13.   display: flex;                  /* flex 布局模型 */
```

```
14.     flex - direction: column;                    /* 垂直布局 */
15.  }
16.  /* 条目盒子 */
17.  .box {
18.     border - bottom: 1rpx solid silver;          /* 1rpx 宽的银色实线下边框 */
19.     margin: 10rpx 20rpx;                         /* 上下外边距 10rpx,左右外边距 20rpx */
20.     padding: 10rpx;                              /* 内边距 */
21.  }
22.  /* 文本样式 */
23.  text{
24.     margin - right:25rpx;
25.  }
```

为了预览已经创建完成的效果,可以临时在 index.js 的 data 中设置 myCard 为 true,看完后再改回 false。

JS(pages/index/index.js)代码片段修改如下:

```
1.  Page({
2.     data: {
3.        myCard:true                                //预览后请改回 false
4.     }
5.  })
```

当前效果如图 8-9 所示。

由图可见,此时首页已经完成创建医疗急救卡前后的布局和样式设计工作。



视频讲解

2 医疗急救卡创建页设计

医疗急救卡创建页用于展示医疗信息表单,主要包含以下内容。

- 出生日期:点击弹出滚动选择器选择年、月、日;
- 医疗状况:多行文本框;
- 医疗笔记:多行文本框;
- 过敏反应:多行文本框;
- 用药:多行文本框;
- 血型:点击弹出滚动选择器选择血型(未知、A、B、O、AB);
- 器官捐赠者:切换是或者否;
- 身高:单行文本输入框,单位为厘米;
- 体重:单行文本输入框,单位为千克;
- 紧急联系人号码:单行文本输入框,输入电话号码。

页面设计如图 8-10 所示。

当前页面计划使用如下组件。

- 页面整体:<form>组件;
- 每条信息:<view>组件,并定义 class='box',根据其排列方向再定义 class 为 row 或 col;
- 信息标签:<label>组件;
- 选择器:<picker>组件;
- 是否开关:<switch>组件;
- 多行文本输入框:<textarea>组件;
- 单行文本输入框:<input>组件;
- 按钮:<button>组件。



图 8-9 首页(已填医疗急救卡)预览效果

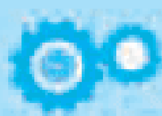


图 8-10 医疗急救卡创建页设计图

医疗急救卡创建页是需要点击首页按钮跳转才可以显示的,为了设计方便,可临时将app.json中pages属性里的form与index路径对调,以确保form页面可以直接显示出来。

WXML(pages/form/form.wxml)代码片段如下:

```
1.  <form bindsubmit='submitForm'>
2.    <view class='row box'>
3.      <label>出生日期</label>
4.      <picker name='date' mode='date' bindchange='dateChange' value='{{date}}'>
5.        <view>{{date}}</view>
6.      </picker>
7.    </view>
8.    <view class='col box'>
9.      <label>医疗状况</label>
10.     <textarea name='ylzk' auto-height value='{{ylzk}}'></textarea>
11.   </view>
12.   <view class='col box'>
13.     <label>医疗笔记</label>
14.     <textarea name='ylbj' auto-height value='{{ylbj}}'></textarea>
15.   </view>
16.   <view class='col box'>
17.     <label>过敏反应</label>
18.     <textarea name='gmfy' auto-height value='{{gmfy}}'></textarea>
19.   </view>
20.   <view class='col box'>
21.     <label>用药</label>
22.     <textarea name='yy' auto-height value='{{yy}}'></textarea>
23.   </view>
24.   <view class='row box'>
25.     <label>血型</label>
26.     <picker name='blood' range='{{bloodItems}}' bindchange='bloodChange' value=
'{{blood}}'>
```



```
27.         <view>{{blood}}型</view>
28.       </picker>
29.     </view>
30.     <view class='row box'>
31.       <label>器官捐赠者</label>
32.       <switch name='qgjz' color='#FF2D55' checked='{{qgjz}}'></switch>
33.     </view>
34.     <view class='row box'>
35.       <label>身高</label>
36.       <input name='height' class='shortInput' type='digit' value='{{height}}'></input>
<text>厘米</text>
37.         <label>体重</label>
38.       <input name='weight' class='shortInput' type='digit' value='{{weight}}'></input>
<text>千克</text>
39.     </view>
40.       <view class='row box'>
41.       <label>紧急联系人号码</label>
42.       <input name='tel' class='longInput' type='number' value='{{tel}}'></input>
43.     </view>
44.     <button form-type='submit'>完成创建</button>
45.       <button bindtap='delMyCard'>删除医疗急救卡</button>
46.   </form>
```

WXSS(pages/form/form.wxss)代码片段如下：

```
1.   /* 表单、多行文本框样式 */
2.   form,textarea{
3.     width: 100%;
4.   }
5.   /* 文本输入框样式 */
6.   .shortInput{
7.     width: 100rpx;
8.   }
9.   .longInput{
10.    width: 200rpx;
11.  }
```

JS(pages/form/form.js)代码片段如下：

```
1.     /**
2.      * 页面的初始数据
3.      */
4.     data: {
5.       date: '2000-01-01',            //出生日期
6.       ylzk: '无',                    //医疗状况
7.       ylbj: '无',                    //医疗笔记
8.       gmfy: '无',                    //过敏反应
9.       yy: '无',                      //用药
10.      blood: 'A',                    //血型
11.      qgjz: false,                   //器官捐赠者
12.      height: '100',                 //身高
13.      weight: '50',                  //体重
14.      bloodItems: ['未知', 'A', 'B', 'O', 'AB'] //血型列表
15.    }
```

当前效果如图 8-11 所示。

由图可见，此时可以显示默认数据内容。



图 8-11 面板 1 预览效果

 8.5 逻辑实现 

8.5.1 尚未创建医疗急救卡的首页的逻辑实现



视频讲解

尚未创建医疗急救卡的首页只有点击“创建医疗急救卡”按钮跳转新页面的功能需要实现。

修改 WXML(pages/index/index.wxml)代码片段如下：

```
1.  <!-- 尚未创建医疗急救卡显示该页面 -->
2.  <view class = 'container' wx:if = '{{!myCard}}'>
3.    <button bindtap = 'goToForm'>创建医疗急救卡</button>
4.  </view>
```

在 JS(pages/index/index.js)中添加 goToForm 函数，代码片段如下：

```
1.  Page({
2.    //打开表单页面
3.    goToForm: function() {
4.      wx.navigateTo({
5.        url: '../form/form',
6.      })
7.    }
8.  })
```

运行效果如图 8-12 所示。

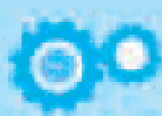


(a) 首页(尚未创建医疗急救卡)



(b) 跳转急救卡创建页

图 8-12 点击“创建医疗急救卡”按钮跳转急救卡创建页

8.5.2 医疗急救卡创建页的逻辑实现



视频讲解

1 更新选择器

修改 WXML 文件(pages/form/form.wxml)的两个< picker >组件,分别为这两个组件追加 bindchange 事件,相关代码如下:

```
1.  <form>
2.    <view class='row box'>
3.      <label>出生日期</label>
4.      <picker name='date' mode='date' bindchange='dateChange' value='{{date}}'>
5.        <view>{{date}}</view>
6.      </picker>
7.    …
8.    <view class='row box'>
9.      <label>血型</label>
10.     <picker name='blood' range='{{bloodItems}}' bindchange='bloodChange' value=
'{{blood}}'>
11.       <view>{{blood}}型</view>
12.     </picker>
13.   </view>
14.   …
15. </view>
```

在 JS 文件(pages/form/form.js)中添加 dateChange 和 bloodChange 函数,代码如下:

```
1.  Page({
2.    //更新日期
3.    dateChange: function(e) {
4.      let value = e.detail.value;          //获得选择的日期
5.      this.setData({ date: value });       //将选项名称更新到 WXML 页面上
6.    },
7.    //更新血型
8.    bloodChange: function(e) {
9.      let i = e.detail.value;              //获得选择的血型序号
10.     this.setData({ blood: this.data.bloodItems[i] });     //将选项名称更新到 WXML 页面上
11.   },
12. })
```

此时可以将滚动选择器修改的内容更新到当前页面,如图 8-13 所示。

2 表单提交



视频讲解

修改 WXML 文件(pages/form/form.wxml)的< form >组件,为该组件追加 bindsubmit 事件,相关代码如下:

```
1.  <form bindsubmit='submitForm'>
2.    …
3.  </form>
```

在 JS 文件(pages/form/form.js)中添加 submitForm 函数,代码如下:

```
1.  Page({
2.    //提交表单
3.    submitForm: function(e) {
```

(a) 初始表单效果　　(b) 更新日期和血型数据后的效果

图 8-13　更新选择器的预览效果

```
4.        //同步保存数据
5.        wx.setStorageSync('myCard', e.detail.value)
6.        //成功后返回首页
7.        wx.navigateBack()
8.      }
9.  })
```

此时填写任意数据，然后单击“完成创建”按钮，调试器 Storage 面板如图 8-14 所示。

图 8-14　表单提交后调试器 Storage 面板的预览效果

由图可见，提交表单后已经成功将数据保存到本地缓存中，其中 Key 名称为 myCard，该名称也可以由开发者自定义。

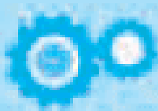


视频讲解

3 删除医疗急救卡

修改 WXML 文件(pages/form/form.wxml)的“删除医疗急救卡”按钮，相关代码如下：

```
1.  <form bindsubmit = 'submitForm'>
2.      …
3.      <button bindtap = 'delMyCard'>删除医疗急救卡</button>
4.  </form>
```



在 JS 文件(pages/form/form.js)中添加 delMyCard 函数,代码如下:

```
1.  Page({
2.    …
3.    //删除医疗急救卡
4.    delMyCard: function() {
5.      //同步删除数据
6.      wx.removeStorageSync('myCard')
7.      //成功后返回首页
8.      wx.navigateBack()
9.    },
10.   …
11. })
```

运行效果如图 8-15 所示。



(a) 填有内容的医疗急救卡



(b) 删除信息并返回首页

图 8-15 删除医疗急救卡的效果

由图可见,此时点击"删除医疗急救卡"按钮将清空缓存数据并返回首页。

8.5.3 已经创建急救卡的首页的逻辑实现



视频讲解

1 读取本地缓存数据

首页需要进行本地缓存的读取,这样才可以正确显示最新的医疗数据。

在 JS 文件(pages/index/index.js)的 onShow 函数中获取本地数据,相关代码如下:

```
1.  Page({
2.    onShow: function() {
3.      //同步获取本地缓存
4.      let myCard = wx.getStorageSync('myCard')
```

```
5.       //更新动态数据
6.       this.setData({ myCard: myCard })
7.     }
8.   })
```

将 WXML 文件(pages/index/index.wxml)的所有医疗数据改为动态数据,相关代码如下:

```
1.  <!-- 急救卡创建完成后显示该页面 -->
2.  <view wx:else>
3.    <view class='row box'>
4.      <label>出生日期</label>
5.      <text>{{myCard.date}}</text>
6.      <label>血型</label>
7.      <text>{{myCard.blood}}型</text>
8.    </view>
9.    <view class='row box'>
10.     <label>身高</label>
11.     <text>{{myCard.height}}厘米</text>
12.     <label>体重</label>
13.     <text>{{myCard.weight}}千克</text>
14.   </view>
15.   <view class='col box'>
16.     <label>医疗状况</label>
17.     <text>{{myCard.ylzk}}</text>
18.   </view>
19.   <view class='col box'>
20.     <label>医疗笔记</label>
21.     <text>{{myCard.ylbj}}</text>
22.   </view>
23.   <view class='col box'>
24.     <label>过敏反应</label>
25.     <text>{{myCard.gmfy}}</text>
26.   </view>
27.   <view class='col box'>
28.     <label>用药</label>
29.     <text>{{myCard.yy}}</text>
30.   </view>
31.   <view class='row box'>
32.     <label>器官捐赠者</label>
33.     <text>{{myCard.qgjz?'是':'否'}}</text>
34.   </view>
35.   <view class='row box'>
36.     <label>紧急联系人</label>
37.     <text>{{myCard.tel}}</text>
38.   </view>
39.   …
40. </view>
```

运行效果如图 8-16 所示。

由图可见,首页已经成功读取到本地缓存中的数据并显示出来。

2 打电话给紧急联系人



视频讲解

修改 WXML 文件(pages/index/index.wxml)中的“打电话给紧急联系人”按钮,为其绑定自定义的 tap 事件,相关代码如下:

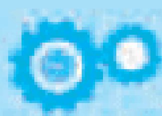
 



图 8-16 显示本地缓存数据

```
1.  <!-- 急救卡创建完成后显示该页面 -->
2.  <view wx:else>
3.    …
4.    <button bindtap = 'makeCall'>打电话给紧急联系人</button>
5.  </view>
```

在 JS 文件(pages/index/index.js)中添加自定义函数 makeCall,相关代码如下:

```
1.  Page({
2.    …
3.    //打电话给紧急联系人
4.    makeCall: function() {
5.      let tel  =  this.data.myCard.tel
6.      wx.makePhoneCall({
7.        phoneNumber: tel
8.      })
9.    },
10.   …
11. })
```

运行效果如图 8-17 所示。

由图可见,开发工具已实现拨打电话的模拟效果。如果使用带有电话卡的真机测试,可以拨打真实电话。

3 编辑医疗急救卡



视频讲解

修改 WXML 文件(pages/index/index.wxml)中的“编辑医疗急救卡”按钮,相关代码如下:

```
1.  <!-- 急救卡创建完成后显示该页面 -->
2.  <view wx:else>
```



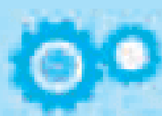
图 8-17　模拟拨打电话的提示

```
3.      …
4.     <button bindtap = 'goToForm'>编辑医疗急救卡</button>
5.   </view>
```

上述代码中的自定义函数 goToForm 在之前已经完成，用于跳转到医疗急救卡创建页。更新 JS 文件（pages/form/form.js）的 onLoad 函数，相关代码如下：

```
1.   Page({
2.     …
3.     onLoad: function(options) {
4.       let myCard  =  wx.getStorageSync('myCard')
5.       //有数据
6.       if (myCard != '') {
7.         this.setData({
8.           date: myCard.date,
9.           ylzk: myCard.ylzk,
10.          ylbj: myCard.ylbj,
11.          gmfy: myCard.gmfy,
12.          yy: myCard.yy,
13.          blood: myCard.blood,
14.          qgjz: myCard.qgjz,
15.          height: myCard.height,
16.          weight: myCard.weight
17.        })
18.      }
19.    },
20.    …
21.  })
```

上述代码表示打开医疗急救卡创建页时先读取一下本地缓存，如果有数据，则显示已经保



存的数据等待新一轮编辑。

运行效果如图 8-18 所示。



图 8-18 显示本地缓存数据的医疗急救卡创建页

由图可见，此时重新打开的医疗急救卡创建页可以显示本地缓存中的数据。



8.6 完整代码展示



8.6.1 应用文件代码展示

app.json 文件的完整代码如下：

```
1.  {
2.    "pages": [
3.      "pages/index/index",
4.      "pages/form/form"
5.    ],
6.    "window": {
7.      "navigationBarBackgroundColor": "#FF2D55",
8.      "navigationBarTitleText": "医疗急救卡"
9.    }
10. }
```

app.wxss 文件的完整代码如下：

```
1.  /* 按钮样式 */
2.  button{
3.    background-color: #FF2D55;            /* 背景颜色为红色 */
4.    color: white;                         /* 字体颜色为白色 */
```

```
5.     margin: 20rpx;                        /* 外边距 */
6.   }
7.   /* 标签样式 */
8.   label {
9.     color: #FF2D55;                       /* 字体颜色为红色 */
10.    margin-right:25rpx;                   /* 右边距 */
11.  }
12.  /* 水平布局 */
13.  .row {
14.    display: flex;                        /* flex 布局模型 */
15.    flex-direction: row;                  /* 水平布局 */
16.  }
17.  /* 垂直布局 */
18.  .col {
19.    display: flex;                        /* flex 布局模型 */
20.    flex-direction: column;               /* 垂直布局 */
21.  }
22.  /* 条目盒子 */
23.  .box {
24.    border-bottom: 1rpx solid silver;     /* 1rpx 宽的银色实线下边框 */
25.    margin: 20rpx;                        /* 外边距 */
26.    padding: 10rpx;                       /* 内边距 */
27.  }
```

8.6.2　页面文件代码展示

1 首页代码展示

WXML 文件(pages/index/index.wxml)的完整代码如下：

```
1.   <!-- pages/index/index.wxml -->
2.   <!-- 尚未创建急救卡显示该页面 -->
3.   <view class='container' wx:if='{{!myCard}}'>
4.     <button bindtap='goToForm'>创建医疗急救卡</button>
5.   </view>
6.   <!-- 急救卡创建完成后显示该页面 -->
7.   <view wx:else>
8.     <view class='row box'>
9.       <label>出生日期</label>
10.      <text>{{myCard.date}}</text>
11.    </view>
12.    <view class='col box'>
13.      <label>医疗状况</label>
14.      <text>{{myCard.ylzk}}</text>
15.    </view>
16.    <view class='col box'>
17.      <label>医疗笔记</label>
18.      <text>{{myCard.ylbj}}</text>
19.    </view>
20.    <view class='col box'>
21.      <label>过敏反应</label>
22.      <text>{{myCard.gmfy}}</text>
23.    </view>
24.    <view class='col box'>
25.      <label>用药</label>
```

```
26.      <text>{{myCard.yy}}</text>
27.    </view>
28.
29.    <view class='row box'>
30.      <label>血型</label>
31.      <text>{{myCard.blood}}型</text>
32.    </view>
33.    <view class='row box'>
34.      <label>器官捐赠者</label>
35.      <text>{{myCard.qgjz?'是':'否'}}</text>
36.    </view>
37.    <view class='row box'>
38.      <label>身高</label>
39.      <text>{{myCard.height}}</text>厘米
40.    </view>
41.    <view class='row box'>
42.      <label>体重</label>
43.      <text>{{myCard.weight}}</text>千克
44.    </view>
45.    <button size='mini' bindtap='goToForm'>编辑医疗急救卡</button>
46.    <button size='mini' bindtap='delMyCard'>删除医疗急救卡</button>
47.  </view>
```

WXSS 文件(pages/index/index.wxss)的完整代码如下：

```
1.  .container{
2.    height: 100vh;                    /* 高 100 视窗，写成 100% 无效 */
3.    display: flex;                    /* flex 布局模型 */
4.    flex-direction: column;           /* 垂直布局 */
5.    align-items: center;              /* 水平方向居中 */
6.    justify-content: center;          /* 垂直方向居中 */
7.  }
```

JS 文件(pages/index/index.js)的完整代码如下：

```
1.  Page({
2.    data: {
3.      myCard:false
4.    },
5.    //打开表单页面
6.    goToForm: function() {
7.      wx.navigateTo({
8.        url: '../form/form',
9.      })
10.   },
11.   //删除医疗急救卡
12.   delMyCard:function(){
13.     wx.removeStorageSync('myCard')
14.     this.setData({ myCard: false })
15.   },
16.   onShow: function() {
17.     let myCard = wx.getStorageSync('myCard')
18.     //有数据
19.     if (myCard != '') {
20.       this.setData({ myCard: myCard })
21.     }
```

```
22.     }
23. })
```

2 医疗急救卡创建页代码展示

WXML 文件(pages/form/form.wxml)的完整代码如下：

```
1.  <form bindsubmit = 'submitForm'>
2.    <view class = 'row box'>
3.      <label>出生日期</label>
4.      <picker name = 'date' mode = 'date' bindchange = 'dateChange' value = '{{date}}'>
5.        <view>{{date}}</view>
6.      </picker>
7.    </view>
8.    <view class = 'col box'>
9.      <label>医疗状况</label>
10.     <textarea name = 'ylzk' auto-height value = '{{ylzk}}'></textarea>
11.   </view>
12.   <view class = 'col box'>
13.     <label>医疗笔记</label>
14.     <textarea name = 'ylbj' auto-height value = '{{ylbj}}'></textarea>
15.   </view>
16.   <view class = 'col box'>
17.     <label>过敏反应</label>
18.     <textarea name = 'gmfy' auto-height value = '{{gmfy}}'></textarea>
19.   </view>
20.   <view class = 'col box'>
21.     <label>用药</label>
22.     <textarea name = 'yy' auto-height value = '{{yy}}'></textarea>
23.   </view>
24.   <view class = 'row box'>
25.     <label>血型</label>
26.     <picker name = 'blood' range = '{{bloodItems}}' bindchange = 'bloodChange' value =
'{{blood}}'>
27.       <view>{{blood}}型</view>
28.     </picker>
29.   </view>
30.   <view class = 'row box'>
31.     <label>器官捐赠者</label>
32.     <switch name = 'qgjz' color = '#FF2D55' checked = '{{qgjz}}'></switch>
33.   </view>
34.   <view class = 'row box'>
35.     <label>身高</label>
36.     <input name = 'height' type = 'digit' value = '{{height}}'></input>厘米
37.   </view>
38.   <view class = 'row box'>
39.     <label>体重</label>
40.     <input name = 'weight' type = 'digit' value = '{{weight}}'></input>千克
41.   </view>
42.   <button form-type = 'submit'>完成创建</button>
43. </form>
```

WXSS 文件(pages/form/form.wxss)的完整代码如下：

```
1.  /*表单、多行文本框样式*/
2.  form,textarea{
3.    width: 100%;
```

```
4.  }
5.  /* 文本输入框样式 */
6.  input{
7.    width: 100rpx;
8.  }
```

JS 文件(pages/form/form.js)的完整代码如下：

```
1.  Page({
2.    data: {
3.      date: '2000-01-01',                    //出生日期
4.      ylzk: '无',                            //医疗状况
5.      ylbj: '无',                            //医疗笔记
6.      gmfy: '无',                            //过敏反应
7.      yy: '无',                              //用药
8.      blood: 'A',                            //血型
9.      qgjz: false,                           //器官捐赠者
10.     height: '100',                         //身高
11.     weight: '50',                          //体重
12.     bloodItems: ['未知', 'A', 'B', 'O', 'AB']     //血型列表
13.   },
14.   //更新日期
15.   dateChange: function(e) {
16.     let value = e.detail.value;            //获得选择的日期
17.     this.setData({ date: value });         //将选项名称更新到 WXML 页面上
18.   },
19.   //更新血型
20.   bloodChange: function(e) {
21.     let i = e.detail.value;                //获得选择的血型序号
22.     this.setData({ blood: this.data.bloodItems[i] });  //将选项名称更新到 WXML 页面上
23.   },
24.   //提交表单
25.   submitForm: function(e) {
26.     console.log(e.detail.value);
27.     //异步保存数据
28.     wx.setStorageSync('myCard', e.detail.value)
29.     //成功后返回首页
30.     wx.navigateBack()
31.   },
32.   onLoad: function (options) {
33.     let myCard = wx.getStorageSync('myCard')
34.     //有数据
35.     if(myCard != '') {
36.       this.setData({
37.         date: myCard.date,
38.         ylzk: myCard.ylzk,
39.         ylbj: myCard.ylbj,
40.         gmfy: myCard.gmfy,
41.         yy: myCard.yy,
42.         blood: myCard.blood,
43.         qgjz: myCard.qgjz,
44.         height: myCard.height,
45.         weight: myCard.weight
46.       })
47.     }
48.   }
49. })
```

第9章



Chapter 9

小程序位置API · 会议邀请函

本章主要使用小程序位置 API 的相关知识制作一款会议邀请函小程序。

本章学习目标

- 掌握获取位置的接口的使用方法；
- 掌握查看位置的接口的使用方法；
- 掌握地图组件控制的系列接口的使用方法。



9.1 项目创建



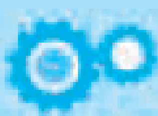


视频讲解

本项目创建选择空白文件夹 invitationDemo，效果如图 9-1 所示。

单击“新建”按钮完成项目创建，然后准备手动创建页面配置文件。

图 9-1 小程序项目填写效果示意图





9.2 页面配置



9.2.1 创建页面文件


视频讲解

项目创建完毕后，在根目录中会生成文件夹 pages 用于存放页面文件。一般来说首页默认命名为 index，表示小程序运行的第一个页面；其他页面名称可以自定义。本项目只需要保留首页(index)即可。

具体操作如下：

(1) 将 app.json 文件内 pages 属性中的“pages/logs/logs”删除，并删除上一行末尾的逗号。

(2) 按快捷键 Ctrl+S 保存当前修改。

9.2.2 删除和修改文件

具体操作如下：

(1) 删除 utils 文件夹及其内部所有内容。

(2) 删除 pages 文件夹下的 logs 目录及其内部所有内容。

(3) 删除 index.wxml 和 index.wxss 中的全部代码。

(4) 删除 index.js 中的全部代码，并且输入关键词 page 找到第二个选项按回车键让其自动补全函数(如图 9-2 所示)。

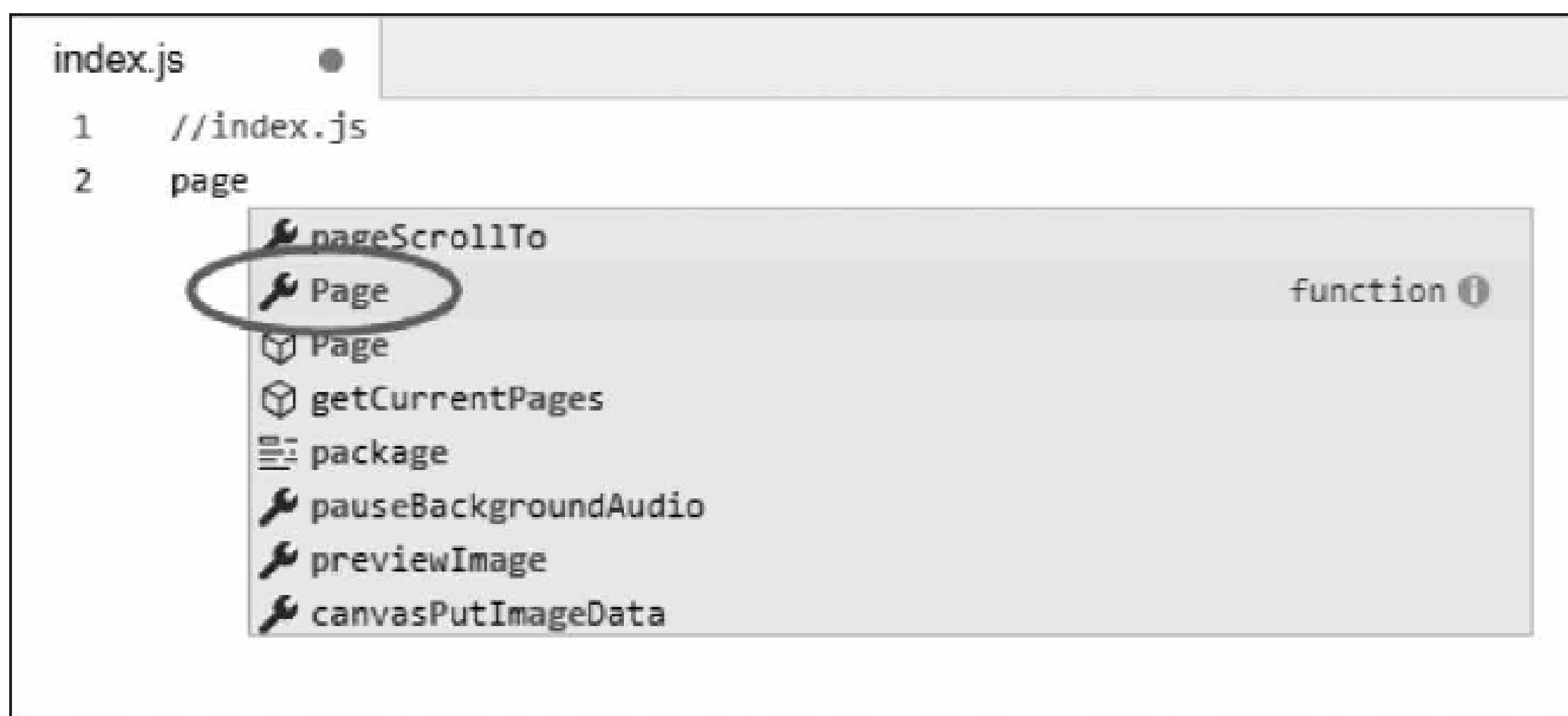


图 9-2 输入关键词创建 Page 函数

(5) 删除 app.wxss 中的全部代码。

(6) 删除 app.js 中的全部代码，并且输入关键词 app 找到第一个选项按回车键让其自动补全函数(如图 9-3 所示)。

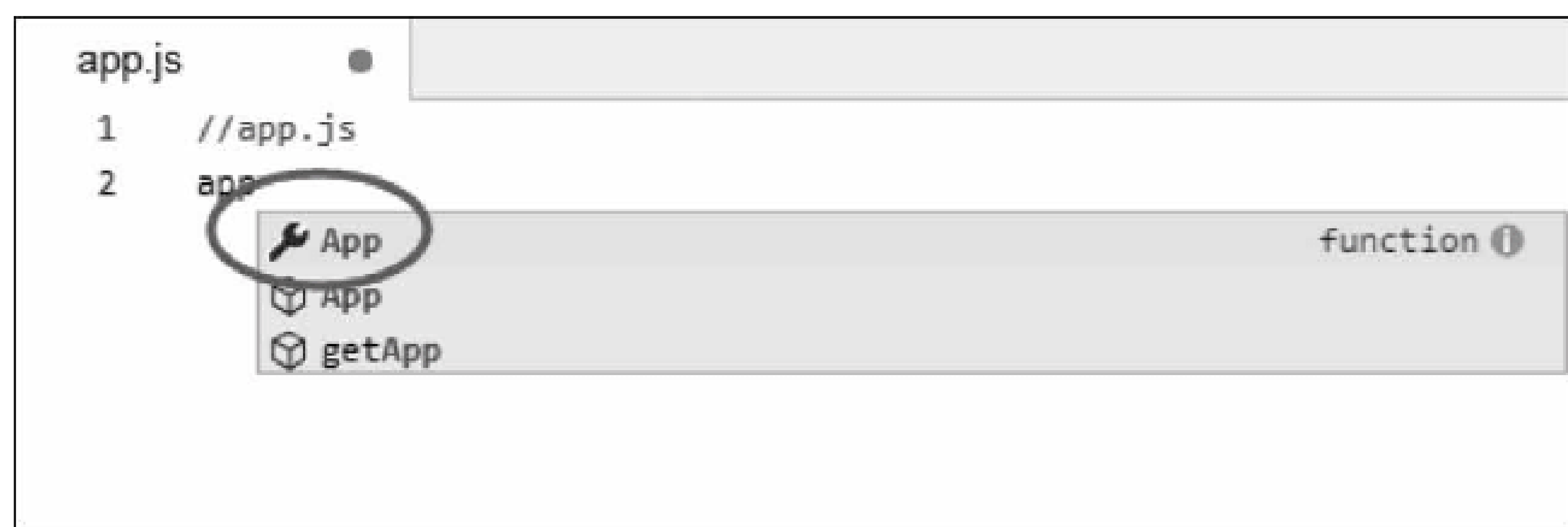


图 9-3 输入关键词创建 App 函数

9.2.3　创建其他文件

接下来创建其他自定义文件，本项目还需要一个文件夹用于存放背景图片和嘉宾头像图标。文件夹名称由开发者自定义（例如 images），单击目录结构左上角的＋号创建文件夹并命名为 images。

本项目用到的背景图片素材如图 9-4 所示。

右击目录结构中的 images 文件夹，选择“硬盘打开”，然后将背景图片文件复制、粘贴进该文件夹。

本项目用到的嘉宾头像图片素材如图 9-5 所示。

右击目录结构中的 images 文件夹，选择“硬盘打开”，然后在其内部新建二级目录 avatar，将图标文件复制、粘贴进该目录。

完成后的目录结构如图 9-6 所示。

本项目的文件配置就全部完成，9.3 节将正式进行页面布局和样式设计。

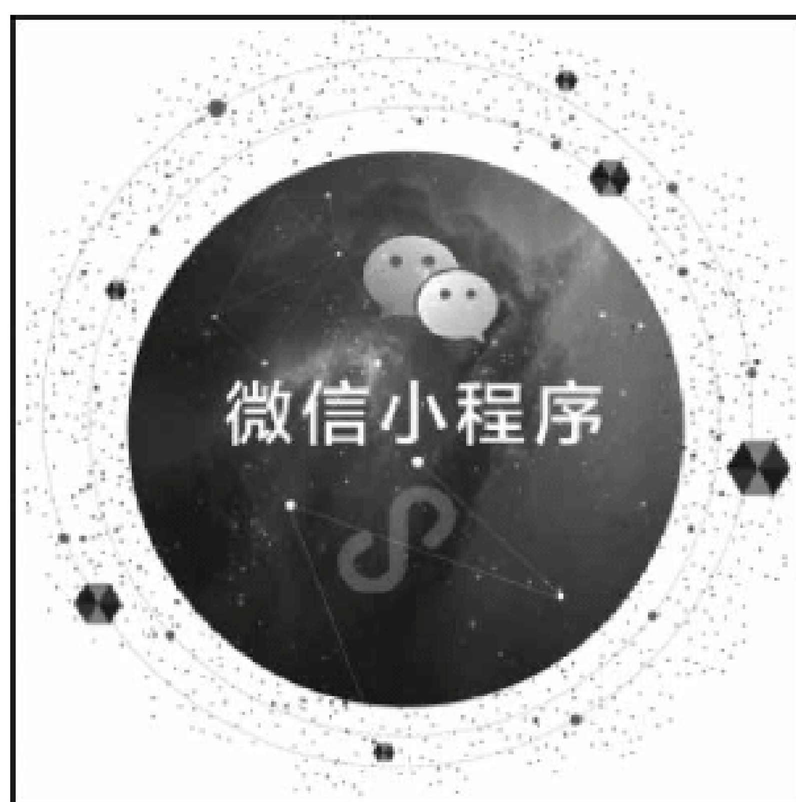
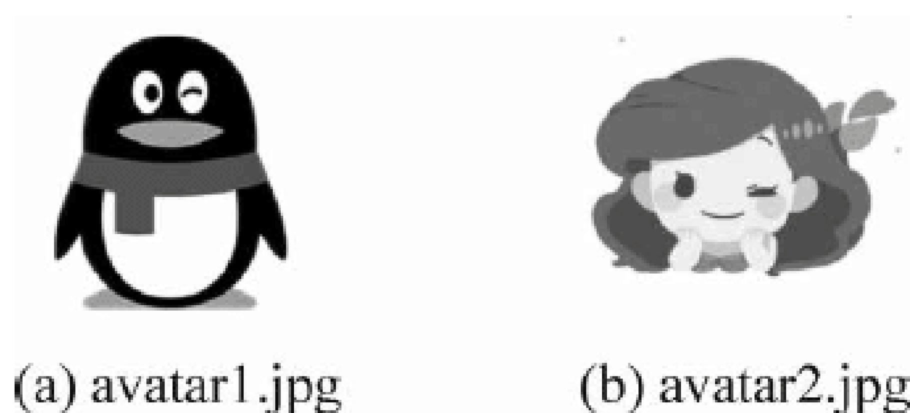


图 9-4　背景图片素材展示（素材来源：网络资源）

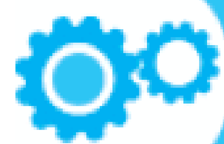


(a) avatar1.jpg　(b) avatar2.jpg

图 9-5　嘉宾头像素材展示（素材来源：网络资源）



图 9-6　页面文件创建完成

 9.3　视图设计

9.3.1　导航栏设计



视频讲解

小程序默认导航栏是黑底白字的效果，因此需要在 app.json 中自定义导航栏标题和背景颜色。更改后的 app.json 文件代码如下：

```
1.  {
2.    "pages": [
3.      "pages/index/index"
4.    ],
5.    "window": {
6.      "navigationBarBackgroundColor": "#00B26A",
```



```
7.        "navigationBarTitleText": "会议邀请函"
8.    }
9.  }
```

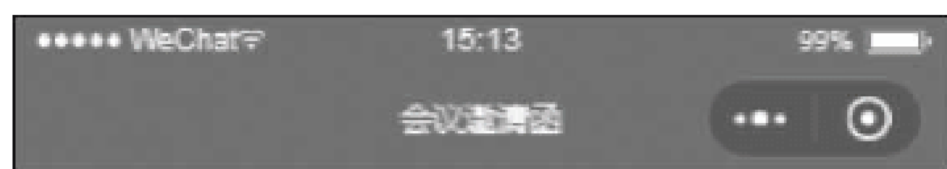


图 9-7　自定义导航栏效果

上述代码可以更改所有页面的导航栏标题文本为“会议邀请函”、背景颜色为绿色(#00B26A)。预览效果如图 9-7 所示。

9.3.2　页面设计



视频讲解

页面上主要以垂直滚动的形式展示会议的各种内容，设计图如图 9-8 所示。

具体区域介绍如下。

- 会议主题：包含背景图片、会议标题、主办方、承办方和协办方信息；
- 活动背景：会议的背景介绍，是纯文本内容；
- 分享嘉宾：以列表的形式显示嘉宾头像和名称；
- 会议地点：显示会议的时间、地点和地图查看效果。

计划使用如下组件。

- 4 个区域：均使用< view >组件，并定义 class='box'；
- 每个区域的标题栏：< view >组件，并定义 class='title'；
- 每个区域的整体内容：< view >组件，并定义 class='content'；
- 每个区域的文本内容：< text >组件；
- 每个区域的图片内容：< image >组件；
- 分享嘉宾区域的列表：< view >组件，并定义 class='bar'；
- 会议地点区域的地图和“查看详情”按钮：< map >组件和< button >组件。



图 9-8　页面设计图

1 整体设计

首先定义 4 个区域的容器(< view >)，WXML(pages/index/index.wxml)代码片段如下：

```
1.  <view class = 'box'>
2.    <view class = 'title'>会议主题</view>
3.    <view class = 'content'>
4.    </view>
5.  </view>
6.  <view class = 'box'>
7.    <view class = 'title'>活动背景</view>
8.    <view class = 'content'>
9.    </view>
10. </view>
11. <view class = 'box'>
12.   <view class = 'title'>分享嘉宾</view>
13.   <view class = 'content'>
```

```
14.     </view>
15.  </view>
16.  <view class = 'box'>
17.     <view class = 'title'>会议地点</view>
18.     <view class = 'content'>
19.     </view>
20.  </view>
```

WXSS(pages/index/index. wxss)代码片段如下：

```
1.   /* 区域整体样式 */
2.   .box {
3.     border: 2rpx solid #00B26A;              /* 边框 */
4.     color: #00B26A;                          /* 字体颜色 */
5.     margin: 15rpx;                           /* 外边距 */
6.     padding: 15rpx;                          /* 内边距 */
7.   }
8.
9.   /* 区域标题样式 */
10.  .title {
11.    font-size: 18pt;                         /* 字体大小 */
12.    font-weight: bold;                       /* 字体加粗 */
13.    padding-bottom: 10rpx;                   /* 底端内边距 */
14.    border-bottom: 2rpx dashed #00B26A;      /* 边框 */
15.  }
16.
17.  /* 区域内容样式 */
18.  .content {
19.    font-size: 12pt;                         /* 字体大小 */
20.    margin: 10rpx 0;                         /* 外边距上下 10rpx,左右 0 */
21.    line-height: 80rpx;                      /* 行间距 */
22.  }
```

当前效果如图 9-9 所示。



图 9-9 页面预览效果

由图可见，目前 4 个区域有了统一的样式效果。

2 会议主题区域设计

会议主题区域需要分别使用< image >和< text >组件追加一个背景图片和一段纯文本描述。

WXML(pages/index/index.wxml)代码片段修改如下：

```
1.  < view class = 'box'>
2.    < view class = 'title'>会议主题</view >
3.    < image src = '/images/banner.jpg' style = 'width:100 % ' mode = 'widthFix'></image >
4.    < view class = 'content'>
5.      < text >全国高校微信小程序开发与实训课程高级研修班 \n 主办方：教育部高等学校计算机类专业教学指导委员会和教育部高等学校软件工程专业教学指导委员会 \n 承办方：集美大学 \n 协办方：腾讯微信事业群和清华大学出版社</text >
6.    </view >
7.  </view >
```

当前效果如图 9-10 所示。

3 活动背景区域设计

活动背景区域只需要使用< text >组件追加一段纯文本内容即可。

WXML(pages/index/index.wxml)代码片段修改如下：

```
1.  < view class = 'box'>
2.    < view class = 'title'>活动背景</view >
3.    < view class = 'content'>
4.      < text > 2018 年腾讯全球合作伙伴大会上，官方公布了一组数据，微信小程序用户量已经达到 2 亿，覆盖行业 200 多个.这一数据显示微信小程序在上线一年的时间内，微信小程序用户确实在呈现指数增长.正是基于此，计划举办本次微信小程序系列课程教学研讨会.</text >
5.    </view >
6.  </view >
```

当前效果如图 9-11 所示。

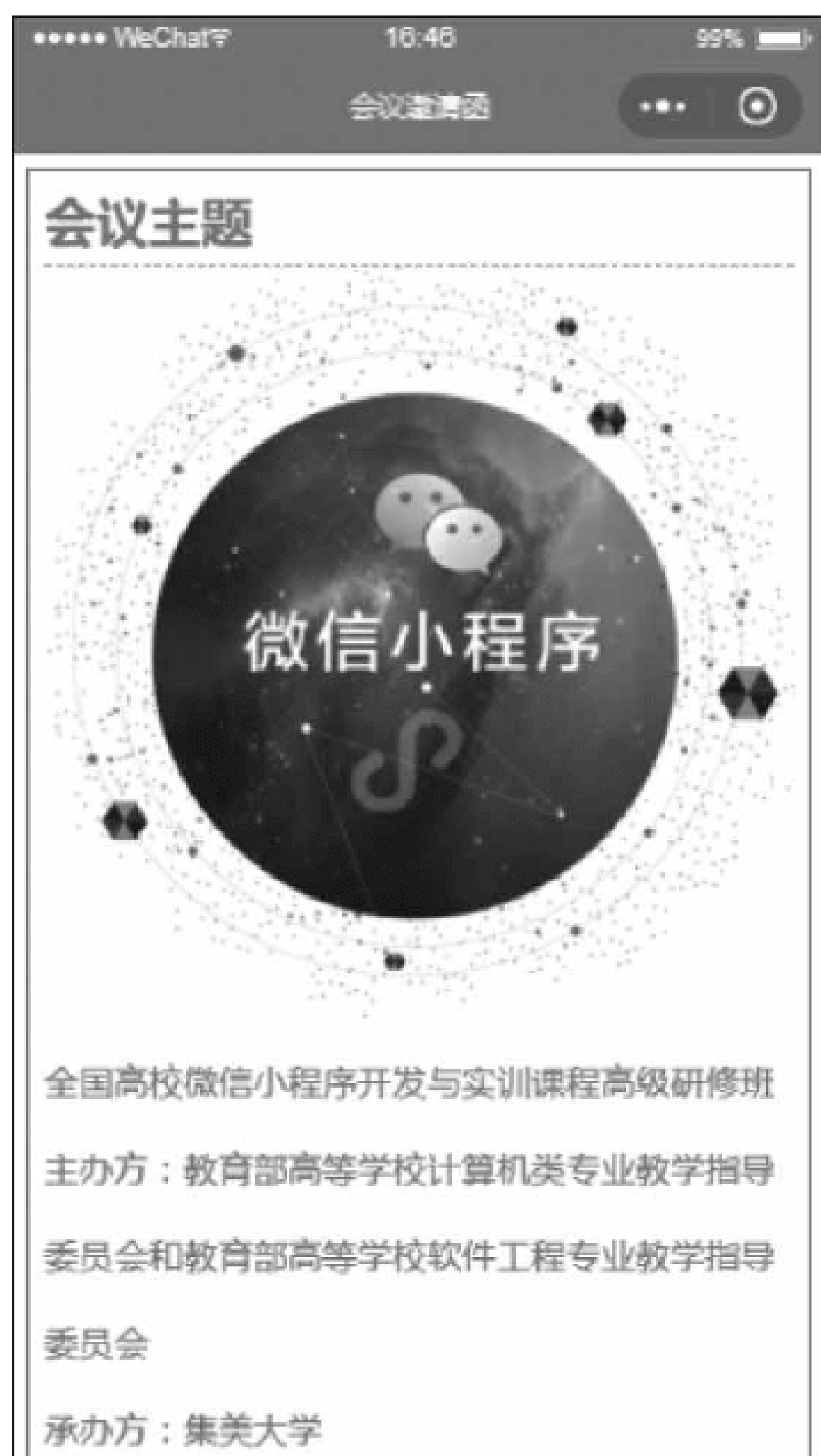


图 9-10　会议主题区域预览效果

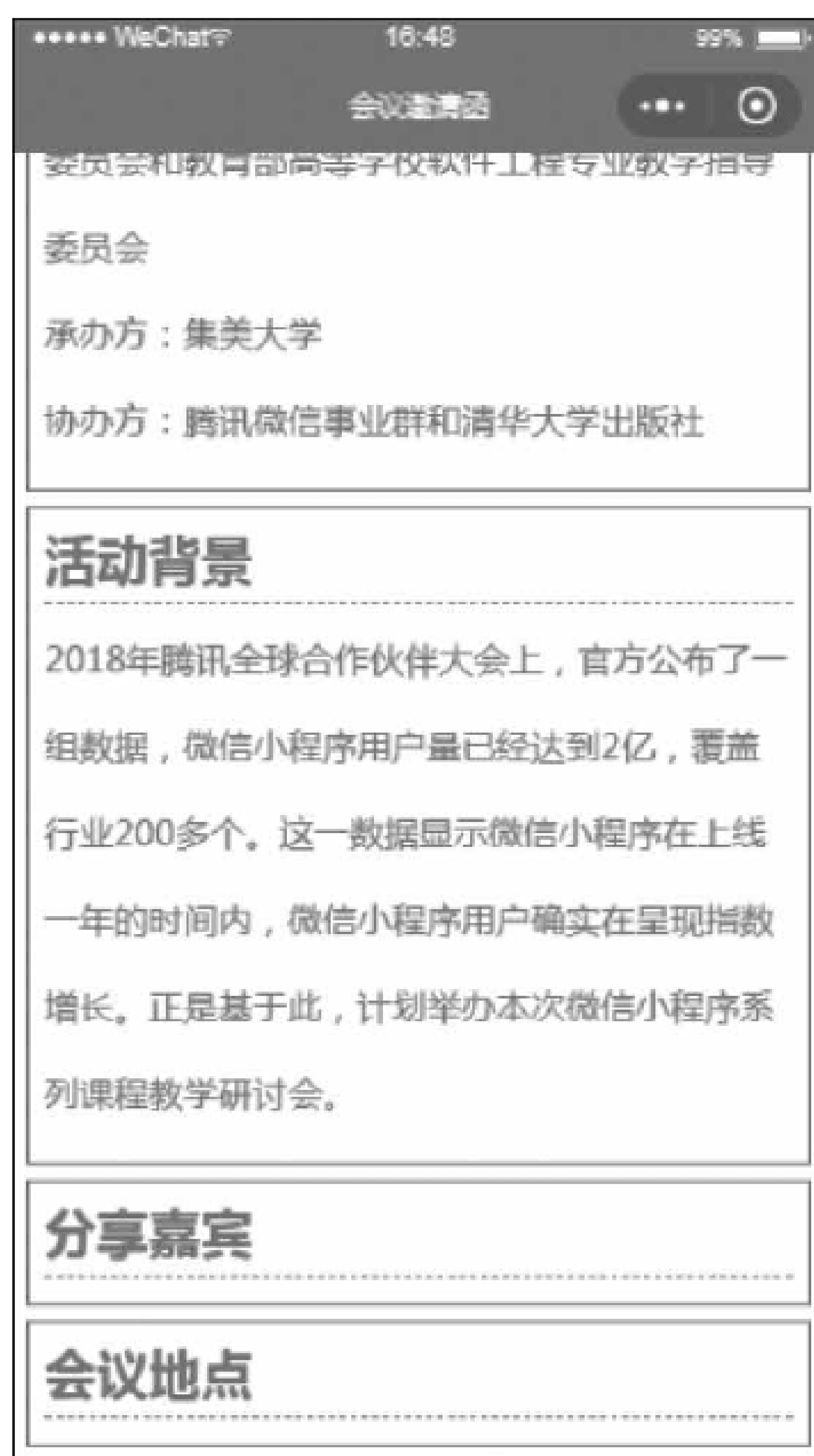


图 9-11　活动背景区域预览效果

4 分享嘉宾区域设计

分享嘉宾区用来显示嘉宾列表,列表中每一行显示一位嘉宾的头像和姓名。这里以只有一位嘉宾为例,WXML(pages/index/index.wxml)代码片段如下:

```
1.  <view class='box'>
2.    <view class='title'>分享嘉宾</view>
3.    <view class='content'>
4.      <view class='bar'>
5.        <image class='avatar' src='/images/avatar/avatar1.jpg'></image>
6.        <text>QQ</text>
7.      </view>
8.    </view>
9.  </view>
```

WXSS(pages/index/index.wxss)代码片段如下:

```
1.  /*分享嘉宾区域:列表单元行样式*/
2.  .bar {
3.    display: flex;                    /*flex模型布局*/
4.    flex-direction: row;              /*水平布局*/
5.    align-items: center;              /*垂直方向居中对齐*/
6.    font-size: 16pt;                  /*字体大小*/
7.  }
8.  /*分享嘉宾区域:头像图标样式*/
9.  .avatar {
10.   width: 200rpx;                    /*宽度*/
11.   height: 200rpx;                   /*高度*/
12.   margin-right: 20rpx;              /*右侧外边距*/
13. }
```

当前效果如图9-12所示。

如果嘉宾人数较多,可以在逻辑实现环节使用wx:for属性循环展示嘉宾列表。

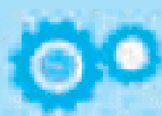
5 会议地点区域设计

会议地点区域主要使用<text>组件实现会议的时间与地点文本描述、使用<map>组件实现地图预览效果以及使用<button>组件实现“查看详情”按钮,WXML(pages/index/index.wxml)代码片段如下:

```
1.  <view class='box'>
2.    <view class='title'>会议地点</view>
3.    <view class='content'>
4.      <text>2019年1月17日-1月20日·厦门</text>
5.      <map></map>
6.      <button>查看详情</button>
7.    </view>
8.  </view>
```

WXSS(pages/index/index.wxss)代码片段如下:

```
1.  /*会议地点区域:地图样式*/
2.  map {
3.    width: 100%;           /*宽度*/
4.    height: 400rpx;        /*高度*/
5.  }
6.  /*会议地点区域:按钮样式*/
```



```
7.  button {
8.    color: white;                    /* 字体颜色 */
9.    background-color: #00B26A;  /* 背景颜色 */
10. }
```

当前效果如图 9-13 所示。



图 9-12　分享嘉宾区域预览效果



图 9-13　会议地点区域预览效果

由图可见，当前地图组件显示的默认地点在北京，后续需要更新为指定的地理位置。此时页面设计就全部完成了，接下来需要进行逻辑实现。



9.4　逻辑实现



9.4.1　更新嘉宾列表



视频讲解

在分享嘉宾区域对<view class='box'>组件添加 wx:for 属性，改写为循环展示列表。

WXML(pages/index/index.wxml)代码片段修改如下：

```
1.  <view class='box'>
2.    <view class='title'>分享嘉宾</view>
3.    <view class='content'>
4.      <view class='bar' wx:for='{{guest}}' wx:key='guest{{index}}'>
5.        <image class='avatar' src='{{item.avatar}}'></image>
6.        <text>{{item.name}}</text>
7.      </view>
8.    </view>
9.  </view>
```

然后在 JS 文件的 data 属性中追加 guest 数组，用于存放嘉宾个人信息。

JS(pages/index/index.js)代码片段修改如下：

```
1.  Page({
2.    /**
3.     * 页面的初始数据
4.     */
5.    data: {
6.     guest: [{
7.           avatar: '/images/avatar/avatar1.jpg',
8.           name: '微信事业部技术人员'
9.         },
10.        {
11.          avatar: '/images/avatar/avatar2.jpg',
12.          name: '周文洁'
13.        }
14.     ]
15. },
16. })
```

开发者可以自行更改嘉宾的人数、头像和名称。运行效果如图 9-14 所示。



图 9-14 更新嘉宾列表效果

由图可见，当前已经可以展示全部嘉宾列表信息。

9.4.2 更新地图位置



视频讲解

假设会议地点在集美大学(坐标：24.579805，118.095086)，修改会议地点区域的< map >组件，为其追加 latitude 和 longitude 属性。

WXML(pages/index/index.wxml)代码片段修改如下：

```
1.  <view class = 'box'>
2.    <view class = 'title'>会议地点</view>
3.    <view class = 'content'>
4.      <text>2019 年 1 月 17 日 - 1 月 20 日·厦门</text>
5.      <map latitude = '{{lat}}' longitude = '{{lon}}'></map>
6.      <button>查看详情</button>
7.    </view>
8.  </view>
```

在 JS 文件的 data 属性中添加坐标数据，JS(pages/index/index.js)代码片段如下：

```
1.  Page({
2.    /**
3.     * 页面的初始数据
4.     */
5.    data: {
6.      lat: 24.579805,
7.      lon: 118.095086,
8.      guest:[ … ]
9.    }
10. })
```

运行效果如图 9-15 所示。



图 9-15　更新地图位置效果

由图可见，当前已经可以展示集美大学附近的地图信息了。

9.4.3　查看地图详情



视频讲解

当用户点击会议地点区域的“查看详情”按钮时可以打开全屏地图进行查看，在真机中还可以进行导航、定位和街景显示。

为< button >组件添加 bindtap 事件，WXML（pages/index/index. wxml）代码片段修改如下：

```
1.  < view class = 'box'>
2.    < view class = 'title'>会议地点</view >
3.    < view class = 'content'>
4.      < text > 2019 年 1 月 17 日 - 1 月 20 日·厦门</text >
5.      < map latitude = '{{lat}}' longitude = '{{lon}}'></map >
6.      < button bindtap = 'showGuide'>查看详情</button >
7.    </view >
8.  </view >
```

在 JS 文件中添加自定义函数 showGuide，JS（pages/index/index. js）代码片段如下：

```
1.  Page({
2.    / **
3.     *  查看位置
4.     * /
5.    showGuide: function() {
6.      var that  =  this
7.      wx.openLocation({
8.        latitude: that.data.lat,
9.        longitude: that.data.lon,
10.     })
11.   },
12. })
```

真机运行效果如图 9-16 所示。

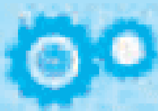


(a) 页面初始效果



(b) 全屏查看地图详情

图 9-16　真机预览效果





(c) 点击右下角按钮的菜单效果



(d) 切换街景模式



(e) 显示建议路线



(f) 跳转高德地图导航

图 9-16 （续）

需要注意的是，如果用户手机中尚未安装第三方地图 App，则只能跳转到应用商店进行下载。

此时会议邀请函小程序就全部完成了，开发者可以根据实际需要更换邀请函的主题颜色、布局和内容。

 9.5 完整代码展示 

app.json 文件的完整代码如下：

```
1.  {
2.    "pages": [
3.      "pages/index/index"
4.    ],
5.    "window": {
6.      "navigationBarBackgroundColor": "#00B26A",
7.      "navigationBarTitleText": "会议邀请函"
8.    }
9.  }
```

WXML 文件(pages/index/index.wxml)的完整代码如下：

```
1.  <view class = 'box'>
2.    <view class = 'title'>会议主题</view>
3.    <image src = '/images/banner.jpg' style = 'width:100%' mode = 'widthFix'></image>
4.    <view class = 'content'>
5.      <text>全国高校微信小程序开发与实训课程高级研修班 \n 主办方: 教育部高等学校计算机类专业教学指导委员会和教育部高等学校软件工程专业教学指导委员会 \n 承办方: 集美大学 \n 协办方: 腾讯微信事业群和清华大学出版社</text>
6.    </view>
7.  </view>
8.  <view class = 'box'>
9.    <view class = 'title'>活动背景</view>
10.   <view class = 'content'>
11.     <text>2018 年腾讯全球合作伙伴大会上,官方公布了一组数据,微信小程序用户量已经达到 2 亿,覆盖行业 200 多个.这一数据显示微信小程序在上线一年的时间内,微信小程序用户确实在呈现指数增长.正是基于此,计划举办本次微信小程序系列课程教学研讨会.</text>
12.   </view>
13. </view>
14. <view class = 'box'>
15.   <view class = 'title'>分享嘉宾</view>
16.   <view class = 'content'>
17.     <view class = 'bar' wx:for = '{{guest}}' wx:key = 'guest{{index}}'>
18.       <image class = 'avatar' src = '{{item.avatar}}' mode = 'widthFix'></image>
19.       <text>{{item.name}}</text>
20.     </view>
21.   </view>
22. </view>
23. <view class = 'box'>
24.   <view class = 'title'>会议地点</view>
25.   <view class = 'content'>
26.     <text>2019 年 1 月 17 日 - 1 月 20 日·厦门</text>
27.     <map latitude = '{{lat}}' longitude = '{{lon}}'></map>
28.     <button bindtap = 'showGuide'>查看详情</button>
29.   </view>
30. </view>
```

WXSS 文件(pages/index/index.wxss)的完整代码如下：

```
1.  /* 区域整体样式 */
```

```
2.  .box {
3.    border: 2rpx solid #00B26A;              /* 边框 */
4.    color: #00B26A;                          /* 字体颜色 */
5.    margin: 15rpx;                           /* 外边距 */
6.    padding: 15rpx;                          /* 内边距 */
7.  }
8.  /* 区域标题样式 */
9.  .title {
10.   font-size: 18pt;                         /* 字体大小 */
11.   font-weight: bold;                       /* 字体加粗 */
12.   padding-bottom: 10rpx;                   /* 底端内边距 */
13.   border-bottom: 2rpx dashed #00B26A;      /* 边框 */
14. }
15. /* 区域内容样式 */
16. .content {
17.   font-size: 12pt;                         /* 字体大小 */
18.   margin: 10rpx 0;                         /* 外边距上下 10rpx,左右 0 */
19.   line-height: 80rpx;                      /* 行间距 */
20. }
21. /* 分享嘉宾区域: 列表单元行样式 */
22. .bar {
23.   display: flex;                           /* flex 模型布局 */
24.   flex-direction: row;                     /* 水平布局 */
25.   align-items: center;                     /* 垂直方向居中对齐 */
26.   font-size: 16pt;                         /* 字体大小 */
27. }
28. /* 分享嘉宾区域: 头像图标样式 */
29. .avatar {
30.   width: 200rpx;                           /* 宽度 */
31.   height: 200rpx;                          /* 高度 */
32.   margin-right: 20rpx;                     /* 右侧外边距 */
33. }
34. /* 会议地点区域: 地图样式 */
35. map {
36.   width: 100%;                             /* 宽度 */
37.   height: 400rpx;                          /* 高度 */
38. }
39. /* 会议地点区域: 按钮样式 */
40. button {
41.   color: white;                            /* 字体颜色 */
42.   background-color: #00B26A;               /* 背景颜色 */
43. }
```

JS 文件(pages/index/index.js)的完整代码如下：

```
1.  Page({
2.    /**
3.     * 页面的初始数据
4.     */
5.    data: {
6.      lat: 24.579805,
7.      lon: 118.095086,
8.      guest: [{
9.          avatar: '/images/avatar/avatar1.jpg',
10.         name: '微信事业部技术人员'
11.       },
```

```
12.         {
13.           avatar: '/images/avatar/avatar2.jpg',
14.           name: '周文洁'
15.         }
16.       ]
17.     },
18.     /**
19.      * 查看位置
20.      */
21.     showGuide: function() {
22.       var that = this
23.       wx.openLocation({
24.         latitude: that.data.lat,
25.         longitude: that.data.lon,
26.       })
27.     },
28. })
```

第10章 Chapter 10



小程序设备API · 指南针

本章主要使用小程序设备 API 的相关知识制作一款指南针小程序。

本章学习目标

- 理解经纬度坐标的含义；
- 了解坐标类别 wgs84 和 gcj02 的区别；
- 使用设备 API 制作指南针小程序。



10.1 项目创建





视频讲解

本项目创建选择空白文件夹 compassDemo，效果如图 10-1 所示。

单击“新建”按钮完成项目创建，然后准备手动创建页面配置文件。

图 10-1　小程序项目填写效果示意图



10.2　页面配置



10.2.1　创建页面文件



视频讲解

项目创建完毕后，在根目录中会生成文件夹 pages 用于存放页面文件。一般来说首页默认命名为 index，表示小程序运行的第一个页面；其他页面名称可以自定义。本项目只需要保留首页(index)即可。

具体操作如下：

(1) 将 app.json 文件内 pages 属性中的“pages/logs/logs”删除，并删除上一行末尾的逗号。

(2) 按快捷键 Ctrl+S 保存当前修改。

10.2.2　删除和修改文件

具体操作如下：

(1) 删除 utils 文件夹及其内部所有内容。

(2) 删除 pages 文件夹下的 logs 目录及其内部所有内容。

(3) 删除 index.wxml 和 index.wxss 中的全部代码。

(4) 删除 index.js 中的全部代码，并且输入关键词 page 找到第二个选项按回车键让其自动补全函数(如图 10-2 所示)。

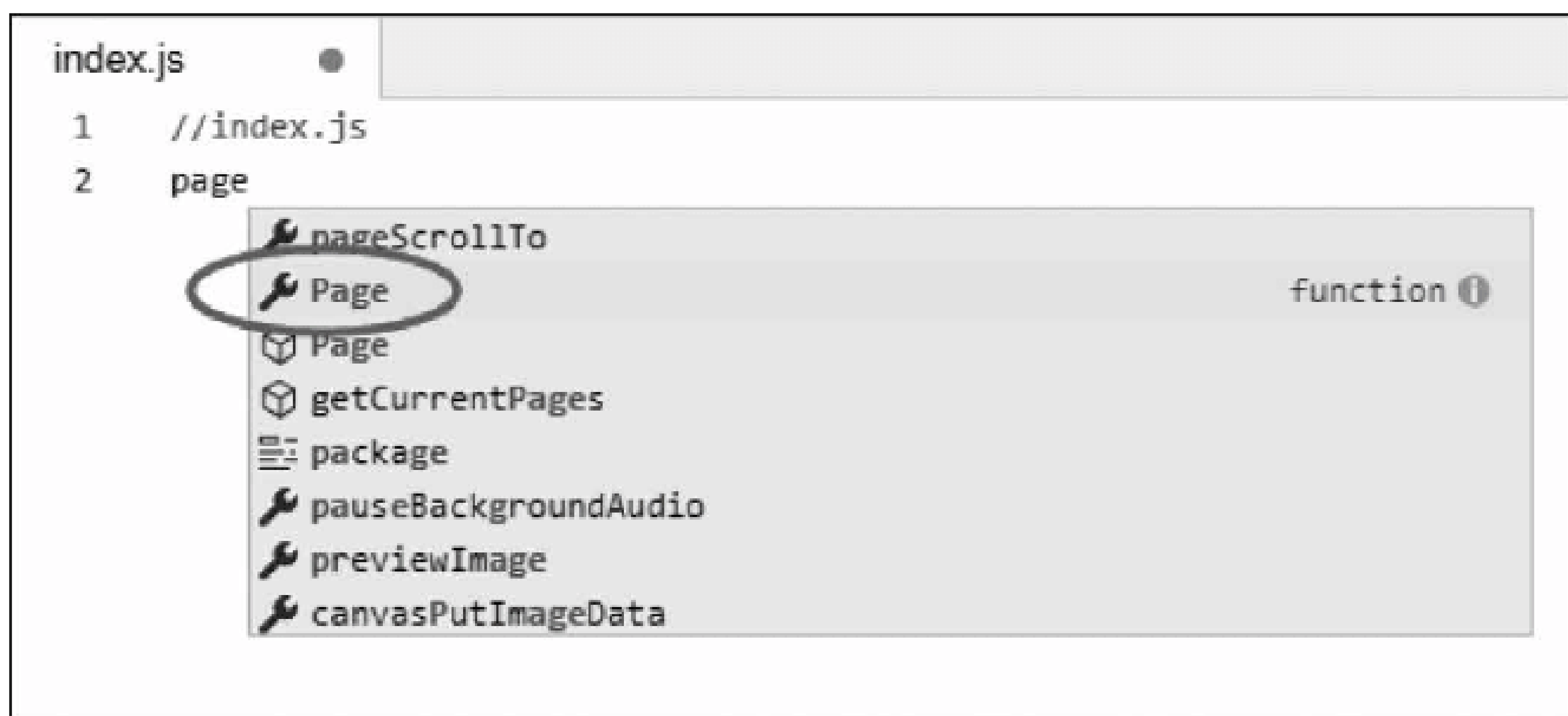


图 10-2　输入关键词创建 Page 函数

(5) 删除 app.wxss 中的全部代码。

(6) 删除 app.js 中的全部代码，并且输入关键词 app 找到第一个选项按回车键让其自动补全函数(如图 10-3 所示)。

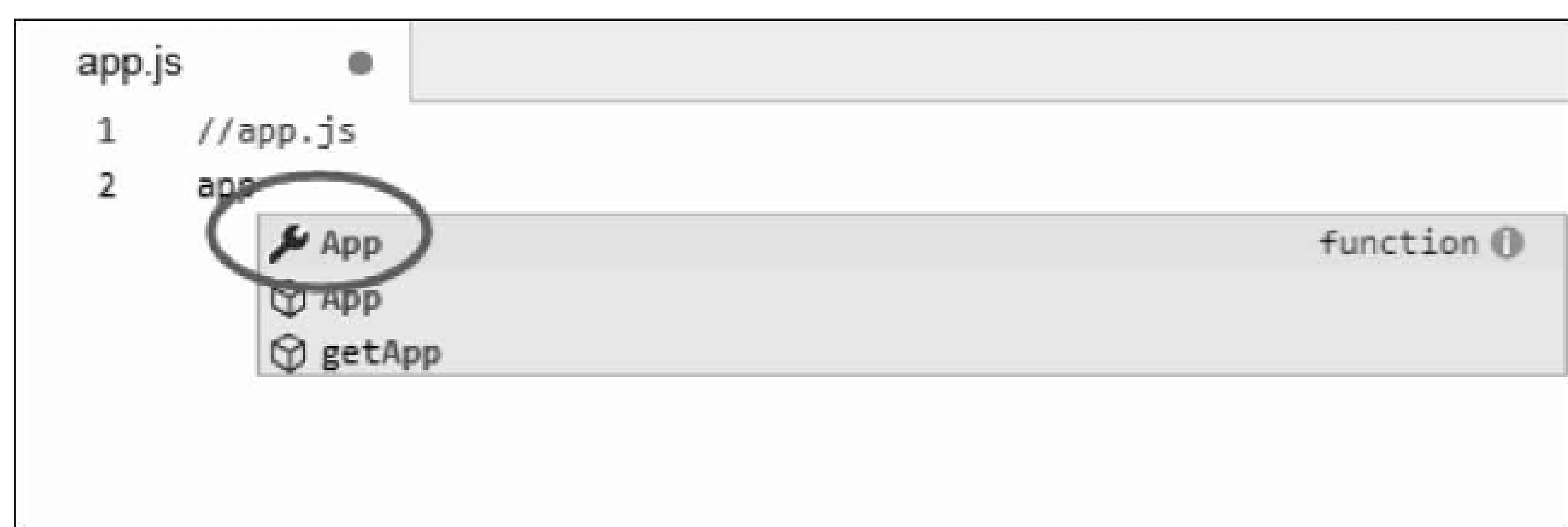


图 10-3　输入关键词创建 App 函数

10.2.3 创建其他文件

接下来创建其他自定义文件，本项目还需要一个文件夹用于存放指南针图片(compass.jpg)。文件夹名称由开发者自定义(例如 images)，单击目录结构左上角的＋号创建文件夹并命名为 images。

本项目用到的背景图片素材如图 10-4 所示。

右击目录结构中的 images 文件夹，选择“硬盘打开”，然后将指南针图片文件复制、粘贴进该文件夹。

全部完成后的目录结构如图 10-5 所示。

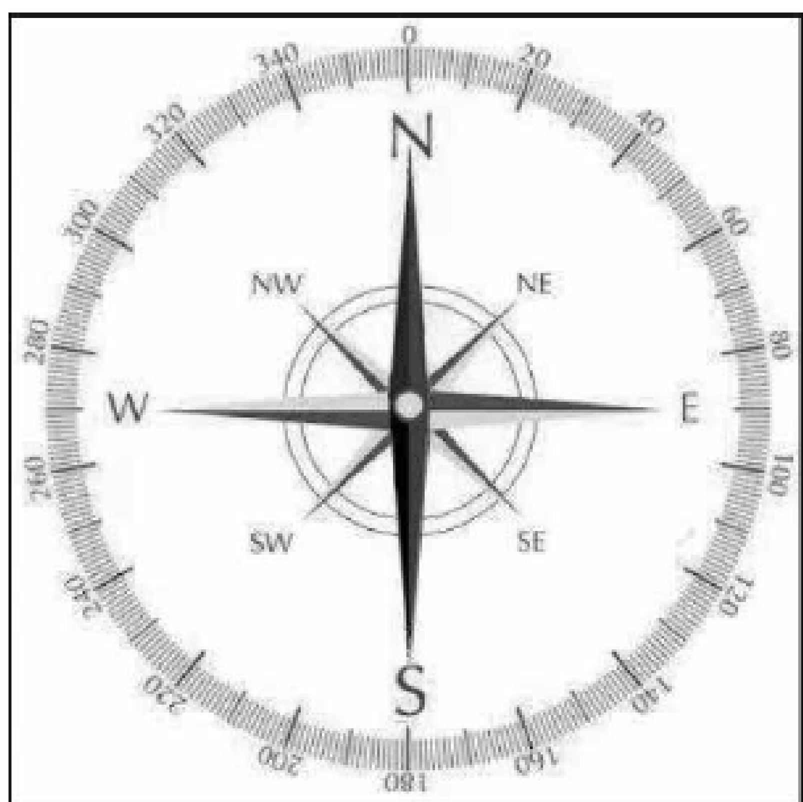
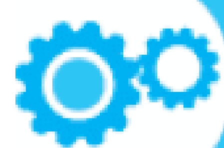


图 10-4　指南针图片素材展示(素材来源：网络资源)



图 10-5　页面文件创建完成

本项目的文件配置就全部完成，10.3 节将正式进行页面布局和样式设计。

 10.3 视图设计 

10.3.1 导航栏设计



视频讲解

小程序默认导航栏是黑底白字的效果，因此需要在 app.json 中自定义导航栏标题和背景颜色。更改后的 app.json 文件代码如下：

```
1.  {
2.    "pages": [
3.      "pages/index/index"
4.    ],
5.    "window": {
6.      "navigationBarBackgroundColor": "#A46248",
7.      "navigationBarTitleText": "我的指南针"
8.    }
9.  }
```

上述代码可以更改所有页面的导航栏标题文本为“我的指南针”，背景颜色为棕色(#A46248)。预览效果如图 10-6 所示。



图 10-6　自定义导航栏效果

10.3.2 页面设计



视频讲解

页面上主要以垂直居中的形式展示指南针图片和状态文字，设计图如图 10-7 所示。

具体区域介绍如下。

- 图片区域：可旋转的指南针图片；
- 文字区域：包括当前方向角度、经纬度和海拔数据。

计划使用如下组件。

- 整体区域：< view >组件，并定义 class='container'；
- 图片区域：< image >组件；
- 文字区域：< view >组件，并定义 class='status'；
- 区域内的文本内容：共 3 行内容，均使用< text >组件。

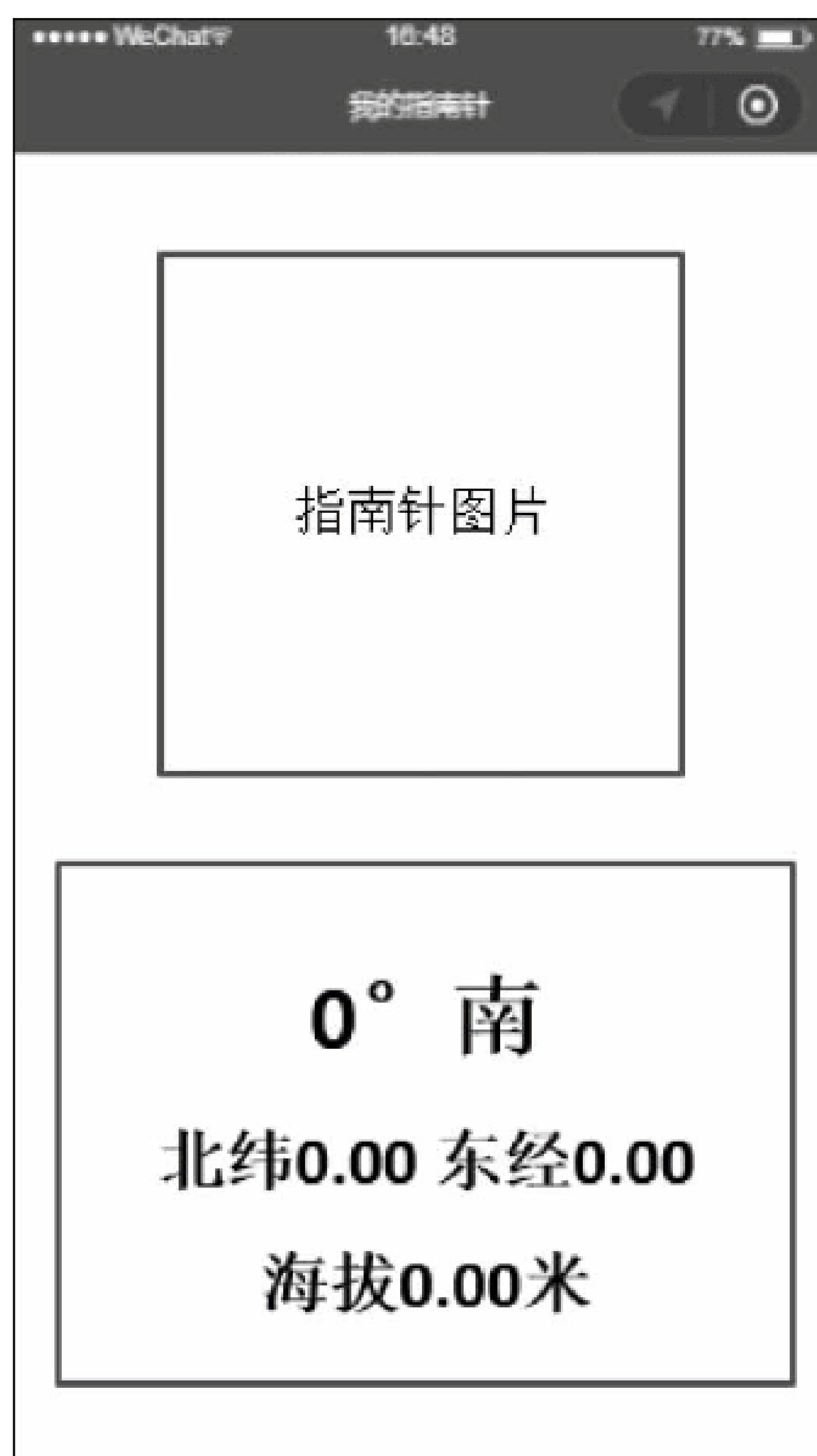


图 10-7 页面设计图

1 整体设计

首先定义整体区域的容器（< view >），WXML（pages/index/index. wxml）代码片段如下：

```
1.  < view class = 'container'>
2.  </view>
```

WXSS(pages/index/index. wxss)代码片段如下：

```
1.  /* 容器整体样式 */
2.  .container{
3.    height: 100vh;                    /* 自适应手机屏幕高度 */
4.    display: flex;                    /* flex 布局模型 */
5.    flex-direction: column;           /* 垂直布局 */
6.    align-items: center;              /* 水平方向居中 */
7.    justify-content: space-around;    /* 调整间隙 */
8.    color: #A46248;                   /* 字体颜色 */
9.  }
```

此时容器中尚未添加具体内容，因此暂时无视图的变化效果。

2 图片区域设计

图片区域需要使用< image >组件添加指南针图片（compass. jpg）。

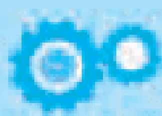
WXML(pages/index/index. wxml)代码片段修改如下：

```
1.  < view class = 'container'>
2.    < image src = '/images/compass.jpg' mode = 'widthFix'></image >
3.  </view>
```

WXSS(pages/index/index. wxss)代码片段如下：

```
1.  /* 图片区域样式 */
2.  image{
3.    width: 80%;                       /* 宽度 */
4.  }
```

当前效果如图 10-8 所示。



3 文字区域设计

文字区域需要使用一个< view >组件包含 3 个< text >组件来完成。

WXML(pages/index/index. wxml)代码片段修改如下：

```
1.  <view class = 'container'>
2.    <image src = '/images/compass.jpg' mode = 'widthFix'></image>
3.    <view class = 'status'>
4.        <text class = 'bigTxt'> 0°南</text>
5.        <text class = 'smallTxt'>北纬 0.00 东经 0.00 </text>
6.        <text class = 'smallTxt'>海拔 0.00 米</text>
7.    </view>
8.  </view>
```

WXSS(pages/index/index. wxss)代码片段如下：

```
1.  /* 文字区域整体样式 */
2.  .status{
3.    display: flex;                        /* flex 布局模型 */
4.    flex-direction: column;               /* 垂直布局 */
5.    align-items: center;                  /* 水平方向居中 */
6.  }
7.  /* 大号字 */
8.  .bigTxt{
9.    font-size: 30pt;                      /* 字体大小 */
10.   margin: 15rpx;                        /* 外边距 */
11. }
12. /* 小号字 */
13. .smallTxt{
14.   font-size: 20pt;                      /* 字体大小 */
15.   margin: 15rpx;                        /* 外边距 */
16. }
```

当前效果如图 10-9 所示。

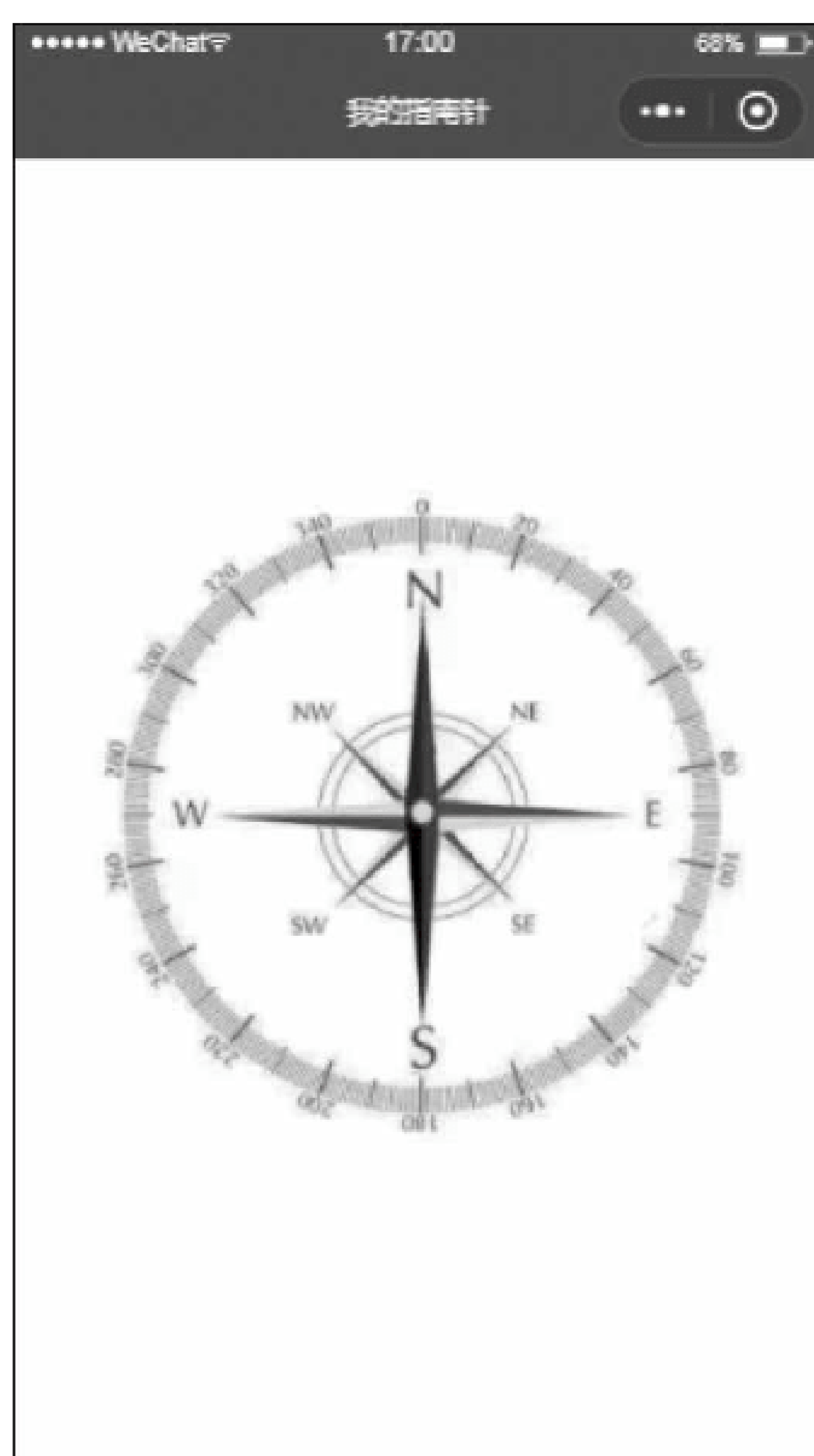


图 10-8 图片区域预览效果

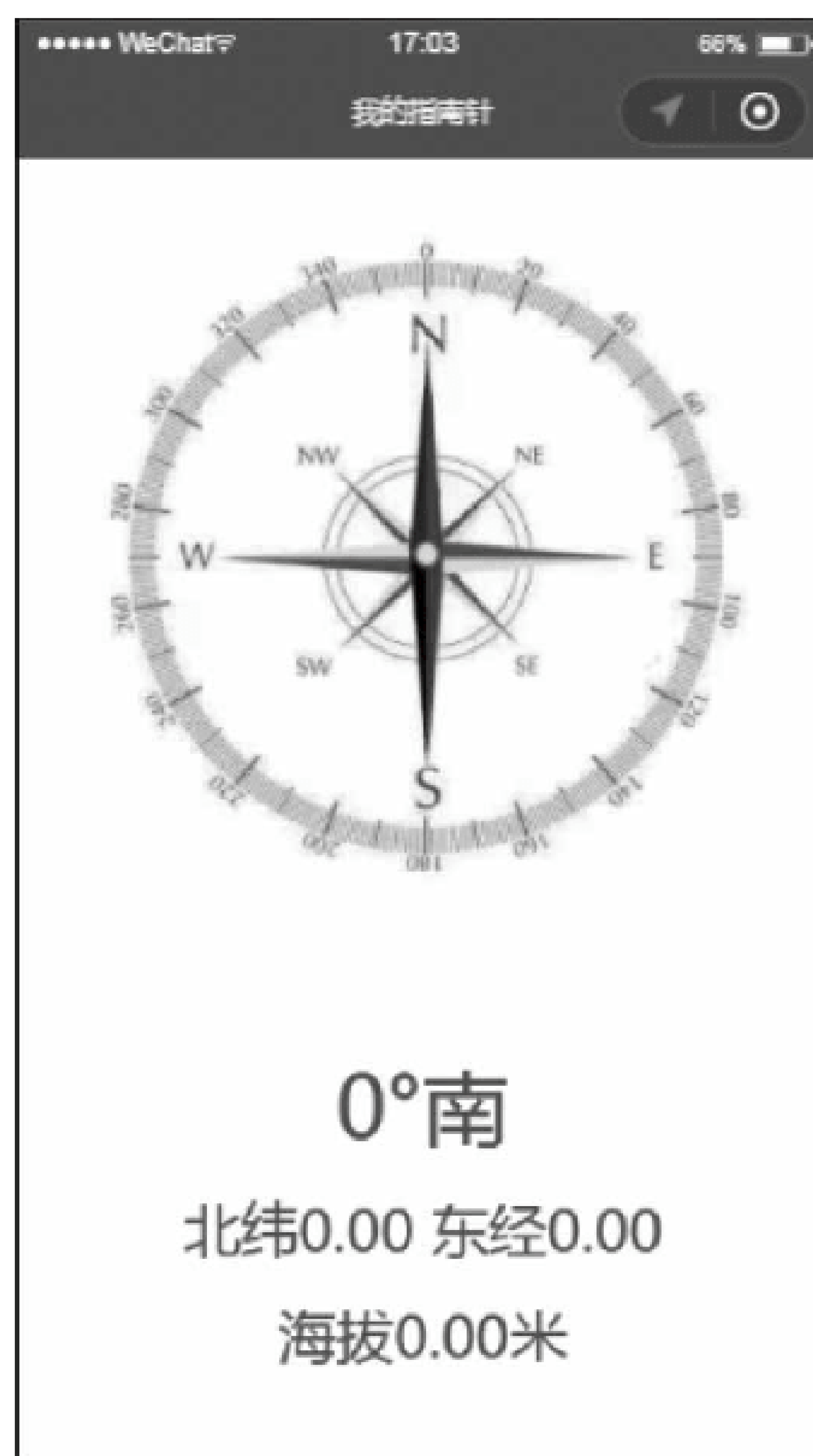
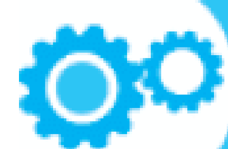


图 10-9 文字区域预览效果

此时页面设计就全部完成了，接下来需要进行逻辑实现。

 10.4 逻辑实现 

10.4.1 指南针旋转动画



视频讲解

修改< image >组件，使用行内样式 style 属性为其添加旋转特效。

WXML(pages/index/index.wxml)代码片段修改如下：

```
1.  < view class = 'container'>
2.    < image src = '/images/compass.jpg' mode = 'widthFix' style = 'transform:rotate({{rotate}}deg);'></image >
3.      < view class = 'status'>
4.        < text class = 'bigTxt'> 0°南</text >
5.        < text class = 'smallTxt'>北纬 0.00 东经 0.00 </text >
6.        < text class = 'smallTxt'>海拔 0.00 米</text >
7.    </view >
8.  </view >
```

上述代码中的 transform:rotate()来源于 CSS3 技术，用于将图片旋转指定角度。例如 transform:rotate(30deg)指的是将图片顺时针旋转 30°。

在 JS 文件的 data 属性中添加 rotate 变量用于存放旋转角度。

JS(pages/index/index.js)代码片段修改如下：

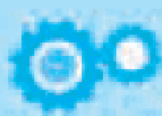
```
1.  Page({
2.    /**
3.     * 页面的初始数据
4.     */
5.    data: {
6.      rotate: 0
7.    },
8.  })
```

开发者可以临时更改旋转角度查看图片旋转效果。

然后在 JS 文件的 onLoad 函数中监听指南针的角度变化。

JS(pages/index/index.js)代码片段修改如下：

```
1.  Page({
2.    /**
3.     * 生命周期函数--监听页面加载
4.     */
5.    onLoad: function(options) {
6.      var that = this;
7.      //监听指南针角度
8.      wx.onCompassChange(function(res) {
9.        //获取度数
10.       let degree = res.direction.toFixed(0)
11.       that.setData({
12.         //计算应偏移度数
13.         rotate: 360 - degree
14.       })
```

```
15.      })
16.    },
17.  })
```

真机预览效果如图 10-10 所示。

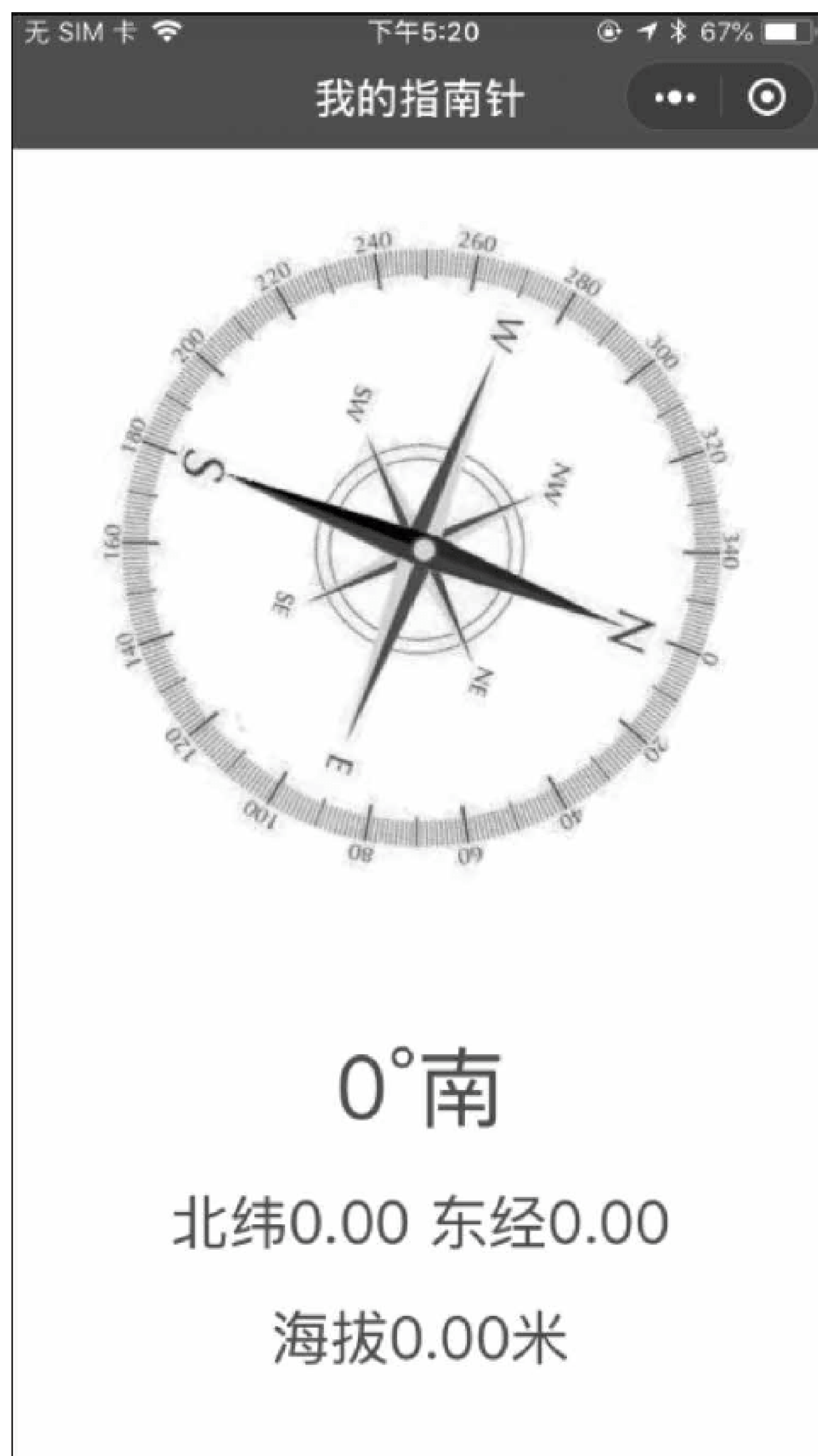


图 10-10　指南针图片旋转效果

由图可见，当前已经可以使用图片显示指南针效果。

10.4.2　更新角度和方向信息



视频讲解

由于文字信息应该根据手机设备的位置不断变化，所以修改< text >组件的内容，将角度和方向信息更新为双花括号描述方式。

WXML(pages/index/index.wxml)代码片段修改如下：

```
1.  <view class='container'>
2.    <image src='/images/compass.jpg' mode='widthFix'></image>
3.    <view class='status'>
4.        <text class='bigTxt'>{{degree}}°{{direction}}</text>
5.        <text class='smallTxt'>北纬 0.00 东经 0.00 </text>
6.        <text class='smallTxt'>海拔 0.00 米</text>
7.    </view>
8.  </view>
```

在 JS 文件的 data 属性中添加相关变量用于存放角度和方向。

JS(pages/index/index.js)代码片段修改如下:

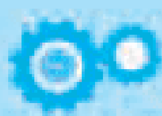
```
1.  Page({
2.    /**
3.     * 页面的初始数据
4.     */
5.    data: {
6.      rotate: 0,
7.      degree: '未知',
8.      direction: ''
9.    },
10. })
```

然后在 JS 文件中添加自定义函数 getDirection,用于更新角度和方向的文字描述,JS(pages/index/index.js)代码片段如下:

```
1.  Page({
2.    /**
3.     * 判断方向
4.     */
5.    getDirection: function(deg) {
6.      let dir = '未知'
7.      if (deg >= 340 || deg <= 20) {
8.        dir = '北'
9.      } else if (deg > 20 && deg < 70) {
10.       dir = '东北'
11.     } else if (deg >= 70 && deg <= 110) {
12.       dir = '东'
13.     } else if (deg > 110 && deg < 160) {
14.       dir = '东南'
15.     } else if (deg >= 160 && deg <= 200) {
16.       dir = '南'
17.     } else if (deg > 200 && deg < 250) {
18.       dir = '西南'
19.     } else if (deg >= 250 && deg <= 290) {
20.       dir = '西'
21.     } else if (deg > 290 && deg < 340) {
22.       dir = '西北'
23.     }
24.     //更新角度和方向
25.     this.setData({
26.       degree: deg,
27.       direction: dir
28.     })
29.   },
30. })
```

修改 onLoad 函数中的指南针角度监听内容,JS(pages/index/index.js)代码片段如下:

```
1.  Page({
2.    /**
3.     * 生命周期函数--监听页面加载
4.     */
5.    onLoad: function(options) {
```



```
6.        var that = this;
7.        //监听指南针角度
8.        wx.onCompassChange(function(res) {
9.          //获取度数
10.         let degree = res.direction.toFixed(0)
11.         //更新方向描述
12.         that.getDirection(degree)
13.         that.setData({
14.           //计算应偏移度数
15.           rotate: 360 - degree
16.         })
17.       })
18.     },
19. })
```

真机预览效果如图 10-11 所示。



图 10-11　更新角度和方向信息效果

由图可见，当前文字区域的第一行内容已经可以随着图片的旋转同步更新了。

10.4.3　更新地理位置信息



视频讲解

由于地理位置信息（经纬度和海拔）也应该根据手机设备的位置不断变化，所以修改< text >组件的内容，将相关信息更新为双花括号描述方式。

WXML(pages/index/index.wxml)代码片段修改如下：

```
1.  <view class = 'container'>
2.    <image src = '/images/compass.jpg' mode = 'widthFix'></image>
3.    <view class = 'status'>
4.        <text class = 'bigTxt'>{{degree}}°{{direction}}</text>
5.        <text class = 'smallTxt'>北纬{{lat}}东经{{lon}}</text>
6.        <text class = 'smallTxt'>海拔{{alt}}米</text>
7.    </view>
8.  </view>
```

在 JS 文件的 data 属性中添加相关变量用于存放经纬度和海拔高度。

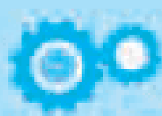
JS(pages/index/index.js)代码片段修改如下：

```
1.  Page({
2.    /**
3.     * 页面的初始数据
4.     */
5.    data: {
6.      rotate: 0,
7.      degree: '未知',
8.      direction: '',
9.      lat: 0,
10.     lon: 0,
11.     alt: 0
12.   },
13. })
```

在 JS 文件的 onLoad 函数中获取地理位置信息，JS(pages/index/index.js)代码片段如下：

```
1.  Page({
2.    /**
3.     * 生命周期函数--监听页面加载
4.     */
5.    onLoad: function(options) {
6.      var that = this;
7.      //获取地理位置
8.      wx.getLocation({
9.        altitude:true,
10.       success: function(res) {
11.         that.setData({
12.           lat: res.latitude.toFixed(2),
13.           lon: res.longitude.toFixed(2),
14.           alt: res.altitude.toFixed(2)
15.         })
16.       },
17.     })
18.     //监听指南针角度
19.     wx.onCompassChange(function(res) {
20.       //代码略
21.     })
22.   },
23. })
```

需要注意的是，腾讯更新了小程序的许可权限，还需要在 app.json 文件中追加以下配置内容方可获取用户地理位置信息。

app.json 文件代码片段如下：

```
1.  {
2.    "pages": [ … 代码略 … ],
3.    "window": { … 代码略 … },
4.    "permission": {
5.      "scope.userLocation": {
6.        "desc": "你的位置信息将用于小程序指南针的效果展示"
7.      }
8.    },
9.  }
```

真机预览效果如图 10-12 所示。

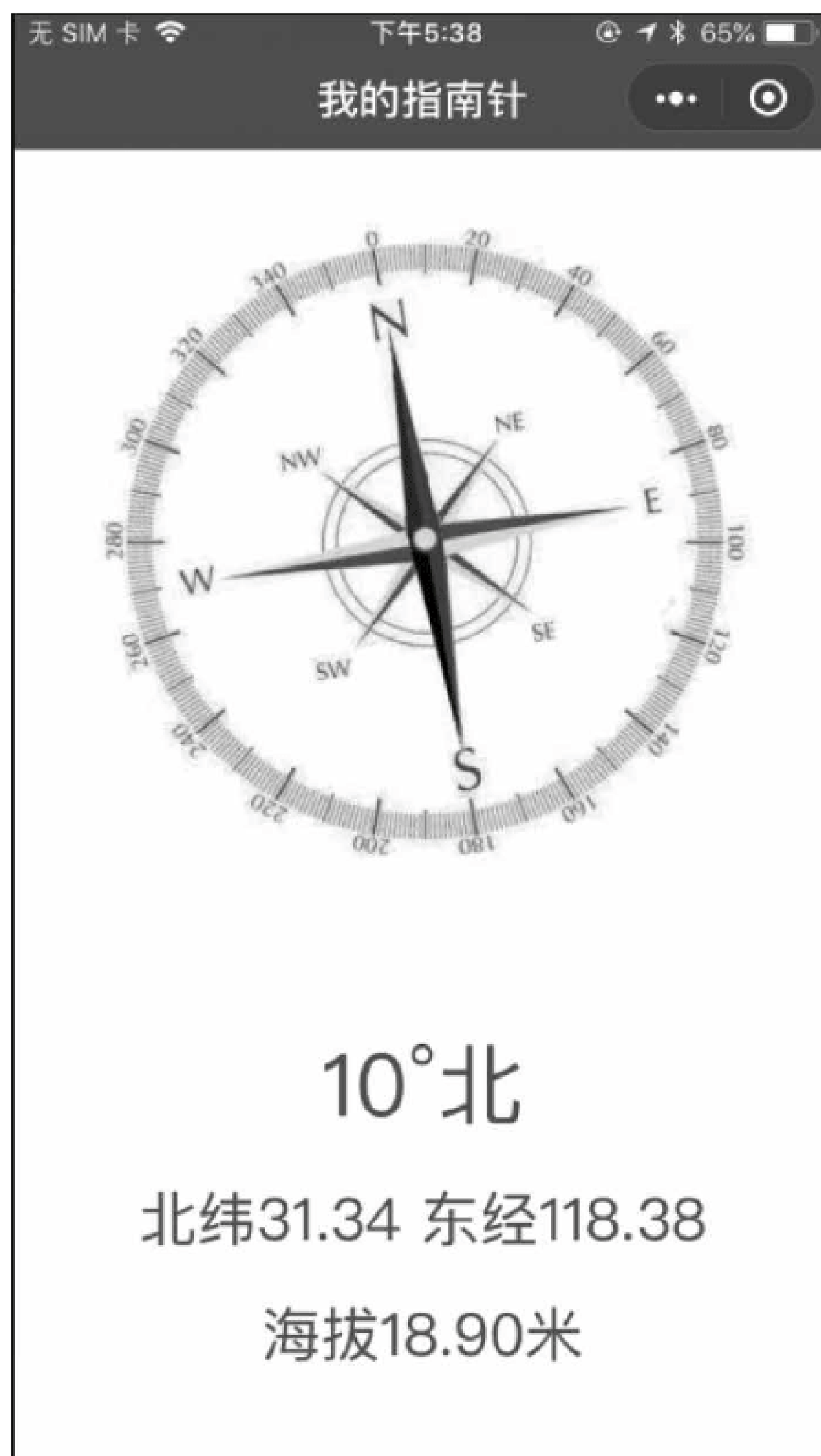


图 10-12　更新地理位置信息效果

由图可见，当前文字区域的全部内容均可以随着手机设备的移动同步更新了。

此时指南针小程序就全部完成了，开发者可以自行更换指南针的图片和样式主题。



10.5　完整代码展示



app.json 文件的完整代码如下：

```
1.  {
2.    "pages": [
3.      "pages/index/index"
4.    ],
```

```
5.     "window": {
6.       "navigationBarBackgroundColor": "#A46248",
7.       "navigationBarTitleText": "我的指南针"
8.     },
9.     "permission": {
10.      "scope.userLocation": {
11.        "desc": "你的位置信息将用于小程序指南针的效果展示"
12.      }
13.    },
14. }
```

WXML 文件(pages/index/index.wxml)的完整代码如下：

```
1.  <view class = 'container'>
2.    <image src = '/images/compass.jpg' mode = 'widthFix' style = 'transform:rotate({{rotate}}deg);'></image>
3.    <view class = 'status'>
4.        <text class = 'bigTxt'>{{degree}}°{{direction}}</text>
5.        <text class = 'smallTxt'>北纬{{lat}} 东经{{lon}}</text>
6.        <text class = 'smallTxt'>海拔{{alt}}米</text>
7.    </view>
8.  </view>
```

WXSS 文件(pages/index/index.wxss)的完整代码如下：

```
1.  /* 容器整体样式 */
2.  .container{
3.    height: 100vh;                        /* 自适应手机屏幕高度 */
4.    display: flex;                        /* flex 布局模型 */
5.    flex-direction: column;               /* 垂直布局 */
6.    align-items: center;                  /* 水平方向居中 */
7.    justify-content: space-around;        /* 调整间隙 */
8.    color: #A46248;                       /* 字体颜色 */
9.  }
10. /* 图片区域样式 */
11. image{
12.   width: 80%;                           /* 宽度 */
13. }
14.
15. /* 文字区域整体样式 */
16. .status{
17.   display: flex;                        /* flex 布局模型 */
18.   flex-direction: column;               /* 垂直布局 */
19.   align-items: center;                  /* 水平方向居中 */
20. }
21. /* 大号字 */
22. .bigTxt{
23.   font-size: 30pt;                      /* 字体大小 */
24.   margin: 15rpx;                        /* 外边距 */
25. }
26. /* 小号字 */
27. .smallTxt{
28.   font-size: 20pt;                      /* 字体大小 */
29.   margin: 15rpx;                        /* 外边距 */
30. }
```

JS 文件(pages/index/index.js)的完整代码如下：

```
1.  Page({
2.    /**
3.     * 页面的初始数据
4.     */
5.    data: {
6.      rotate: 0,
7.      degree: '未知',
8.      direction: '',
9.      lat: 0,
10.     lon: 0,
11.     alt: 0
12.   },
13.   /**
14.    * 判断方向
15.    */
16.   getDirection: function(deg) {
17.     let dir = '未知'
18.     if (deg >= 340 || deg <= 20) {
19.       dir = '北'
20.     } else if (deg > 20 && deg < 70) {
21.       dir = '东北'
22.     } else if (deg >= 70 && deg <= 110) {
23.       dir = '东'
24.     } else if (deg > 110 && deg < 160) {
25.       dir = '东南'
26.     } else if (deg >= 160 && deg <= 200) {
27.       dir = '南'
28.     } else if (deg > 200 && deg < 250) {
29.       dir = '西南'
30.     } else if (deg >= 250 && deg <= 290) {
31.       dir = '西'
32.     } else if (deg > 290 && deg < 340) {
33.       dir = '西北'
34.     }
35.     //更新角度和方向
36.     this.setData({
37.       degree: deg,
38.       direction: dir
39.     })
40.   },
41.   /**
42.    * 生命周期函数--监听页面加载
43.    */
44.   onLoad: function(options) {
45.     var that = this;
46.     //获取地理位置
47.     wx.getLocation({
48.       success: function(res) {
49.         that.setData({
50.           lat: res.latitude.toFixed(2),
51.           lon: res.longitude.toFixed(2),
52.           alt: res.altitude.toFixed(2)
53.         })
54.       },
```

```
55.     })
56.     //监听指南针角度
57.     wx.onCompassChange(function(res) {
58.       //获取度数
59.       let degree = res.direction.toFixed(0)
60.       //更新方向描述
61.       that.getDirection(degree)
62.       that.setData({
63.         //计算应偏移度数
64.         rotate: 360 - degree
65.       })
66.     })
67.   },
68. })
```

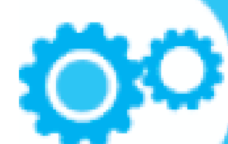
第11章  Chapter 11

小程序界面API · 手绘时钟

本章主要使用小程序界面 API 中绘图的相关知识制作一款手绘时钟小程序。

本章学习目标

- 掌握画布组件(< canvas >)的基础用法；
- 掌握绘制矩形、路径和文本的方法；
- 掌握颜色与样式的设置方法；
- 掌握 setInterval 函数的用法，设置每秒刷新画面。



11.1 项目创建





视频讲解

本项目创建选择空白文件夹 clockDemo，效果如图 11-1 所示。

单击“新建”按钮完成项目创建，然后准备手动修改页面配置文件。

图 11-1 小程序项目填写效果示意图



11.2　页面配置



11.2.1　创建页面文件


视频讲解

项目创建完毕后，在根目录中会生成文件夹 pages 用于存放页面文件。一般来说首页默认命名为 index，表示小程序运行的第一个页面；其他页面名称可以自定义。本项目只需要首页（index）即可。

具体操作如下：

（1）将 app.json 文件内 pages 属性中的“pages/logs/logs”删除，并删除上一行末尾的逗号。

（2）按快捷键 Ctrl+S 保存当前修改。

11.2.2　删除和修改文件

具体操作如下：

（1）删除 utils 文件夹及其内部所有内容。

（2）删除 pages 文件夹下的 logs 目录及其内部所有内容。

（3）删除 index.wxml 和 index.wxss 中的全部代码。

（4）删除 index.js 中的全部代码，并且输入关键词 page 找到第二个选项按回车键让其自动补全函数（如图 11-2 所示）。

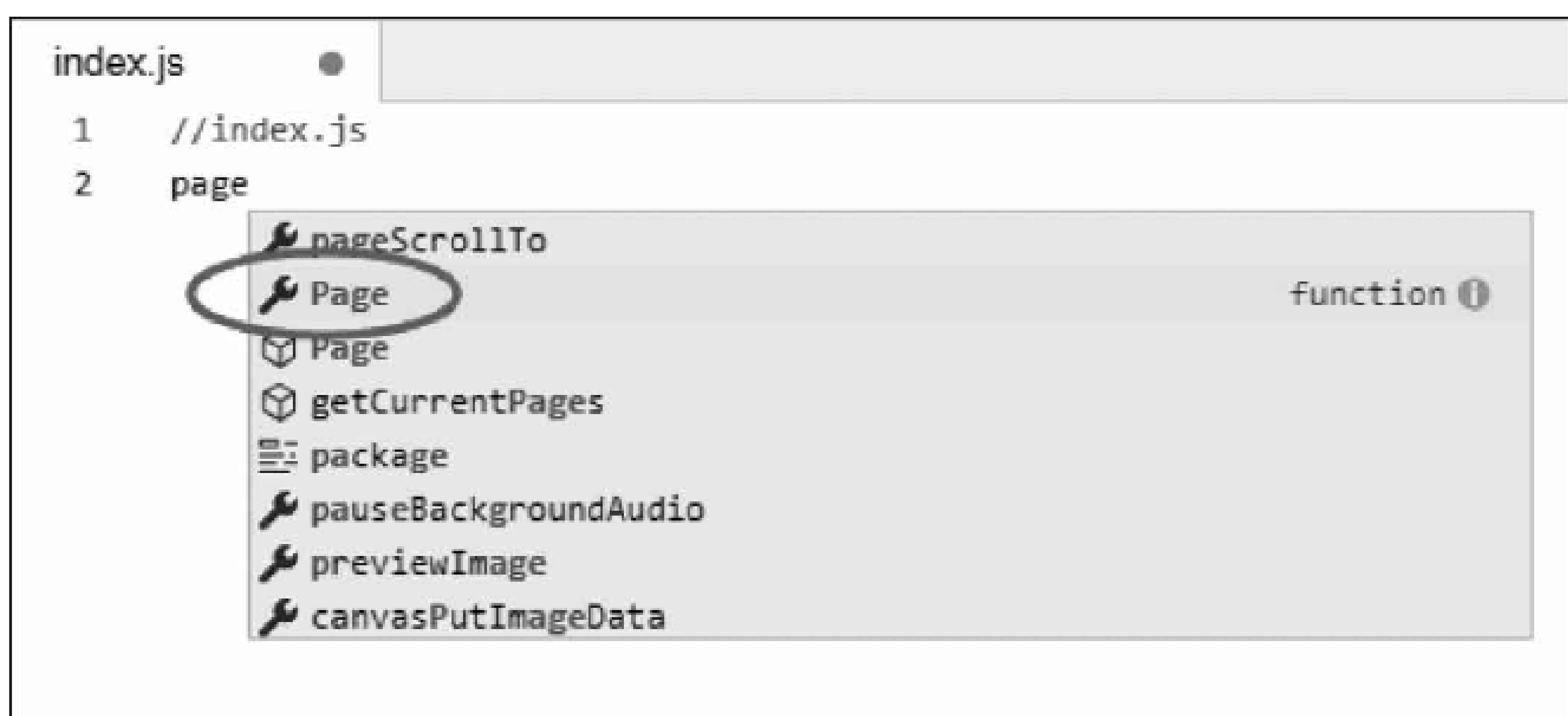


图 11-2　输入关键词创建 Page 函数

（5）删除 app.wxss 中的全部代码。

（6）删除 app.js 中的全部代码，并且输入关键词 app 找到第一个选项按回车键让其自动补全函数（如图 11-3 所示）。

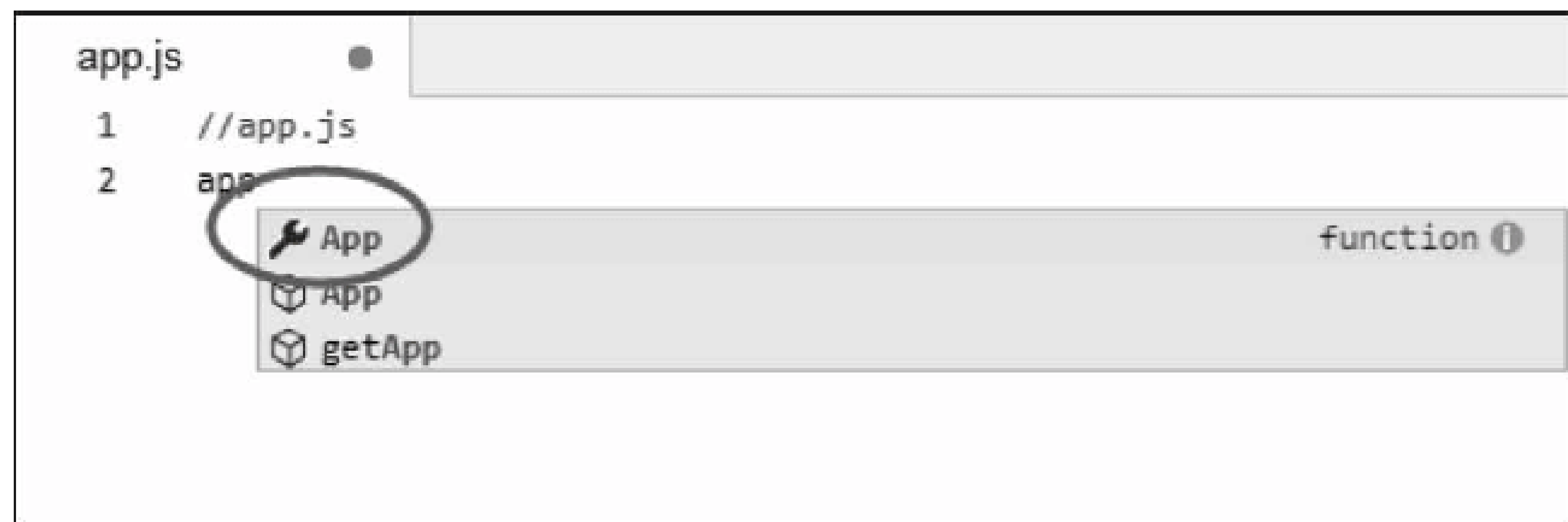
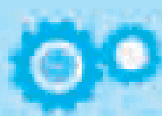


图 11-3　输入关键词创建 App 函数

本项目的文件配置就全部完成，11.3 节将正式进行页面布局和样式设计。



11.3 视图设计



11.3.1 导航栏设计



视频讲解

小程序默认导航栏是黑底白字的效果，因此只需要在 app.json 中自定义导航栏标题。

更改后的 app.json 文件代码如下：

```
1.  {
2.    "pages": [
3.      "pages/index/index"
4.    ],
5.    "window": {
6.      "navigationBarTitleText": "我的时钟"
7.    }
8.  }
```

上述代码可以更改所有页面的导航栏标题文本为“我的时钟”。



图 11-4　自定义导航栏效果

预览效果如图 11-4 所示。

11.3.2 页面设计



视频讲解

页面上主要以垂直居中的形式展示标题、手绘时钟和数字电子时钟，设计图如图 11-5 所示。

具体区域介绍如下。

- 画布区域：手绘时钟；
- 文字区域：标题和数字电子时钟。

计划使用如下组件。

- 整体区域：< view >组件，并定义 class='container'；
- 手绘时钟区域：< canvas >组件；
- 数字电子时钟区域：< text >组件。

1 整体设计

首先定义整体区域的容器(< view >)，WXML(pages/index/index.wxml)代码片段如下：

```
1.  < view class = 'container'>
2.  </view>
```

WXSS(pages/index/index.wxss)代码片段如下：

```
1.  /* 整体容器样式 */
2.  .container{
3.    height: 100vh;                    /* 高度 */
4.    display: flex;                    /* flex 布局模型 */
5.    flex-direction: column;           /* 垂直布局 */
6.    align-items: center;              /* 水平方向居中 */
7.    justify-content: space-around;    /* 调整内容间隙 */
8.  }
```

此时容器中尚未添加具体内容，因此暂时无视图的变化效果。

2 标题区域设计

标题区域需要使用< text >组件实现。

WXML(pages/index/index.wxml)代码片段修改如下：

```
1.  < view class = 'container'>
2.    < text > My Clock </text >
3.  </view >
```

WXSS(pages/index/index.wxss)代码片段如下：

```
1.  /* 文本样式 */
2.  text{
3.    font - size: 40pt;                    /* 字号大小 */
4.    font - weight: bold;                  /* 字体加粗 */
5.  }
```

当前效果如图 11-6 所示。

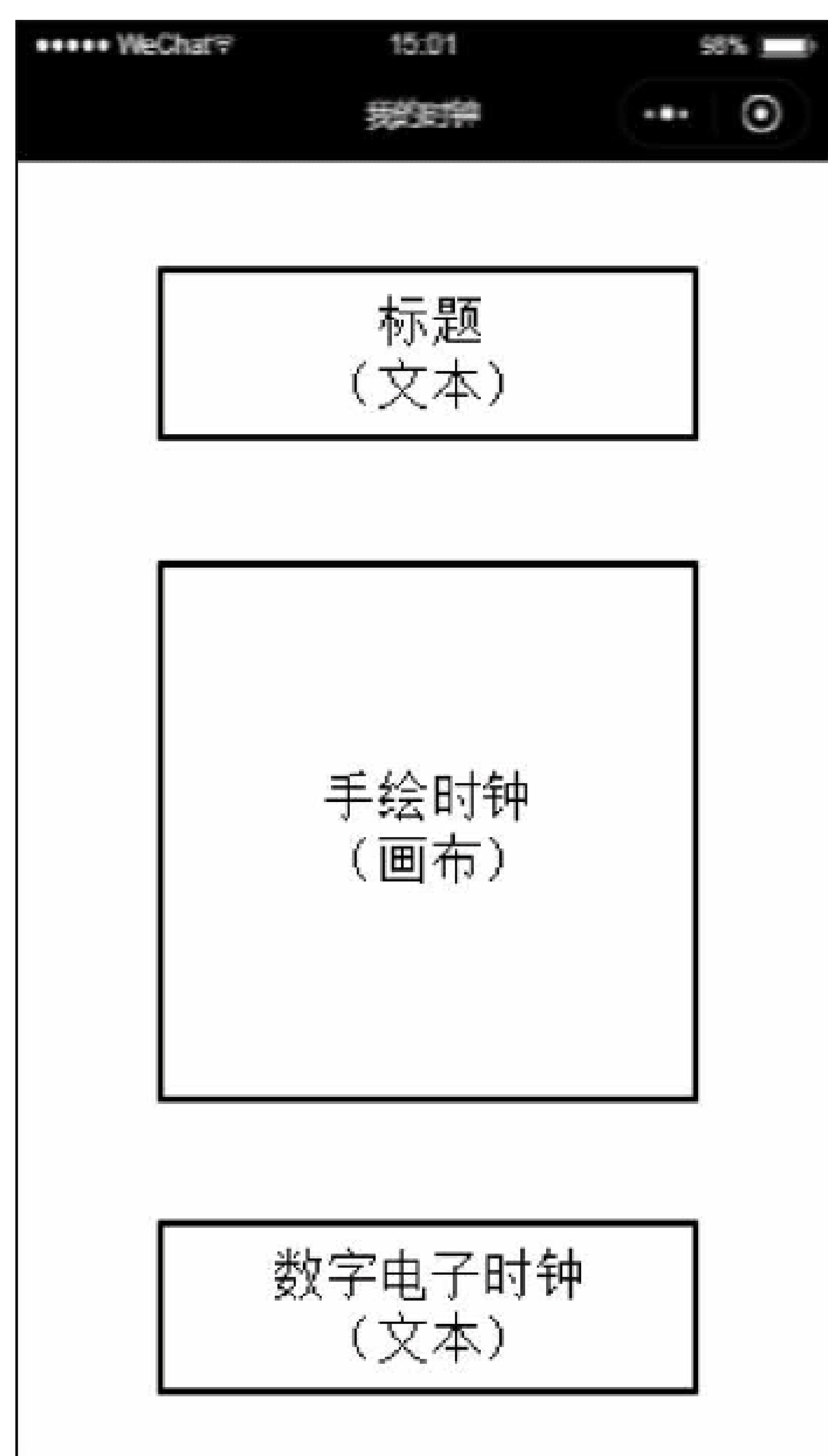


图 11-5 页面设计图

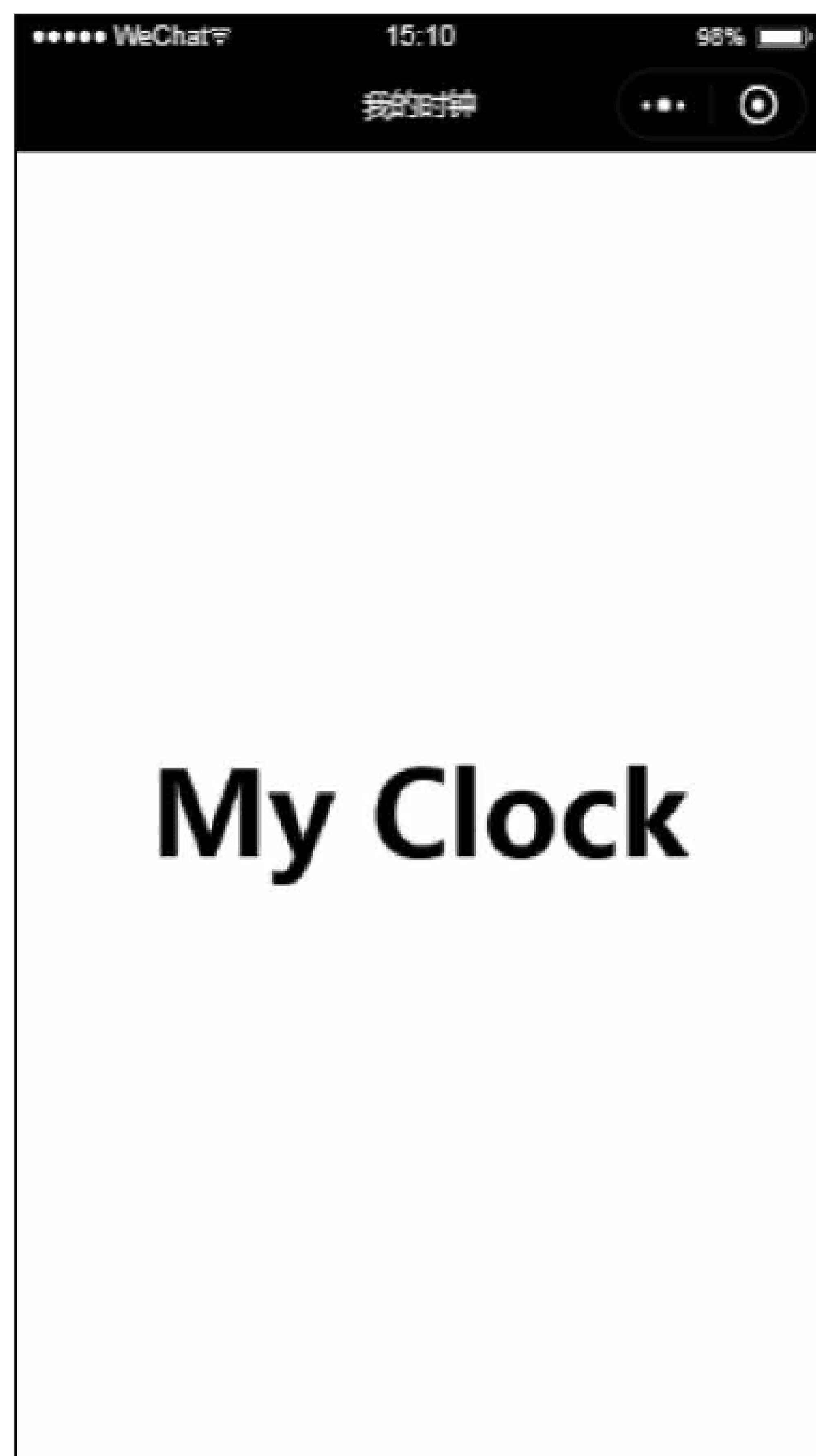


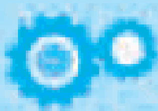
图 11-6 标题区域预览效果

3 手绘时钟区域设计

手绘时钟区域需要使用< canvas >组件实现。

WXML(pages/index/index.wxml)代码片段修改如下：

```
1.  < view class = 'container'>
2.    < text > My Clock </text >
3.    < canvas canvas - id = 'clockCanvas'></canvas >
4.  </view >
```

WXSS(pages/index/index.wxss)代码片段如下：

```
1.  /* 画布样式 */
2.  canvas{
3.    width: 600rpx;                    /* 宽度 */
4.    height: 600rpx;                   /* 高度 */
5.    border:1rpx solid red;            /* 临时属性,用于查看画布边框 */
6.  }
```

当前效果如图 11-7 所示。

在确认画布位置和尺寸后，可以删除或注释掉 WXSS 样式中临时添加的 border 属性。

4 数字电子时钟区域设计

数字电子时钟区域需要使用<text>组件实现。

WXML(pages/index/index.wxml)代码片段修改如下：

```
1.  <view class='container'>
2.    <text>My Clock</text>
3.    <canvas canvas-id='clockCanvas'></canvas>
4.    <text>12:00:00</text>
5.  </view>
```

此时文本中显示的时间为临时效果，后续将替换为真实时间信息。

当前效果如图 11-8 所示。

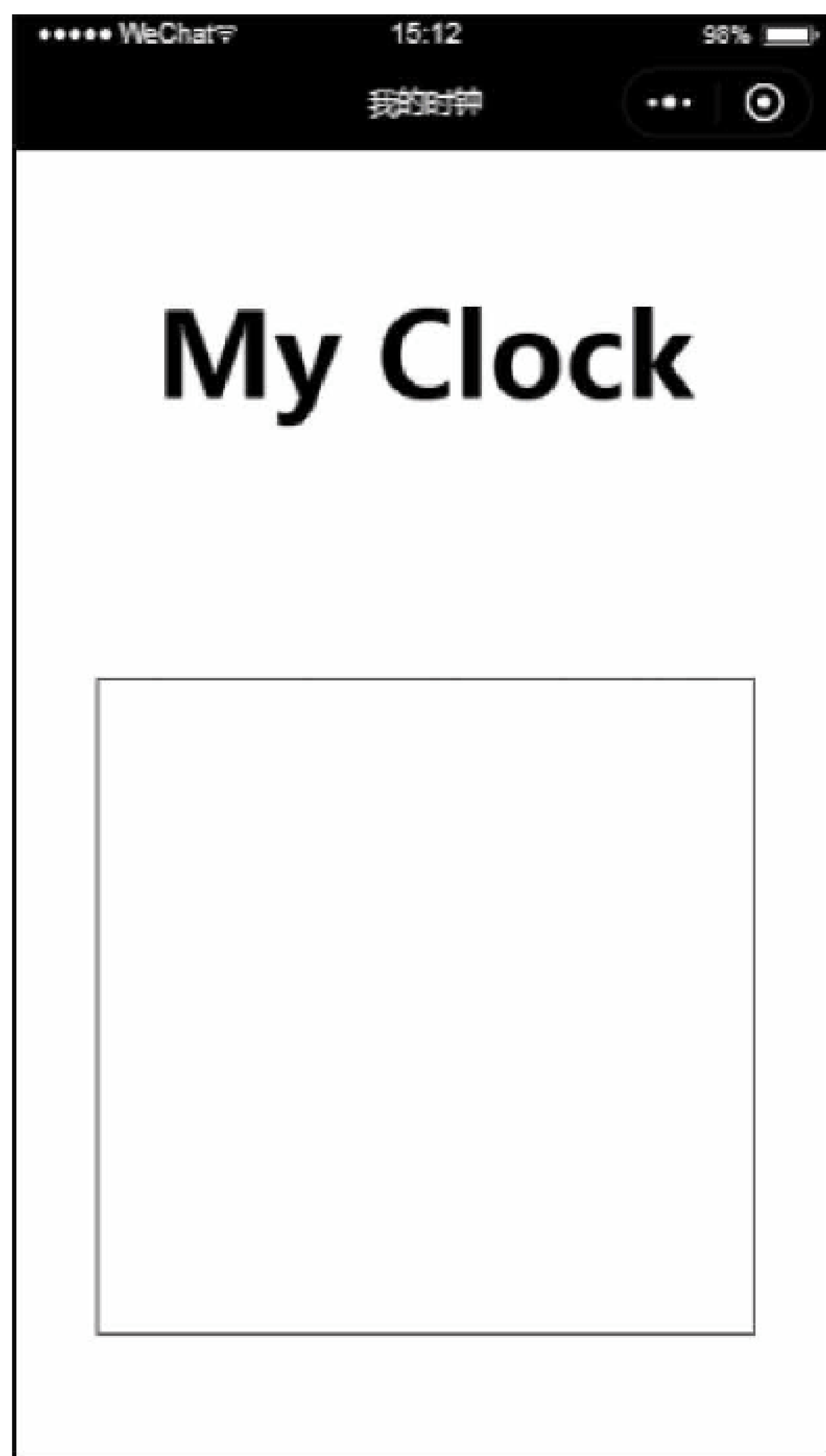


图 11-7　手绘时钟区域预览效果

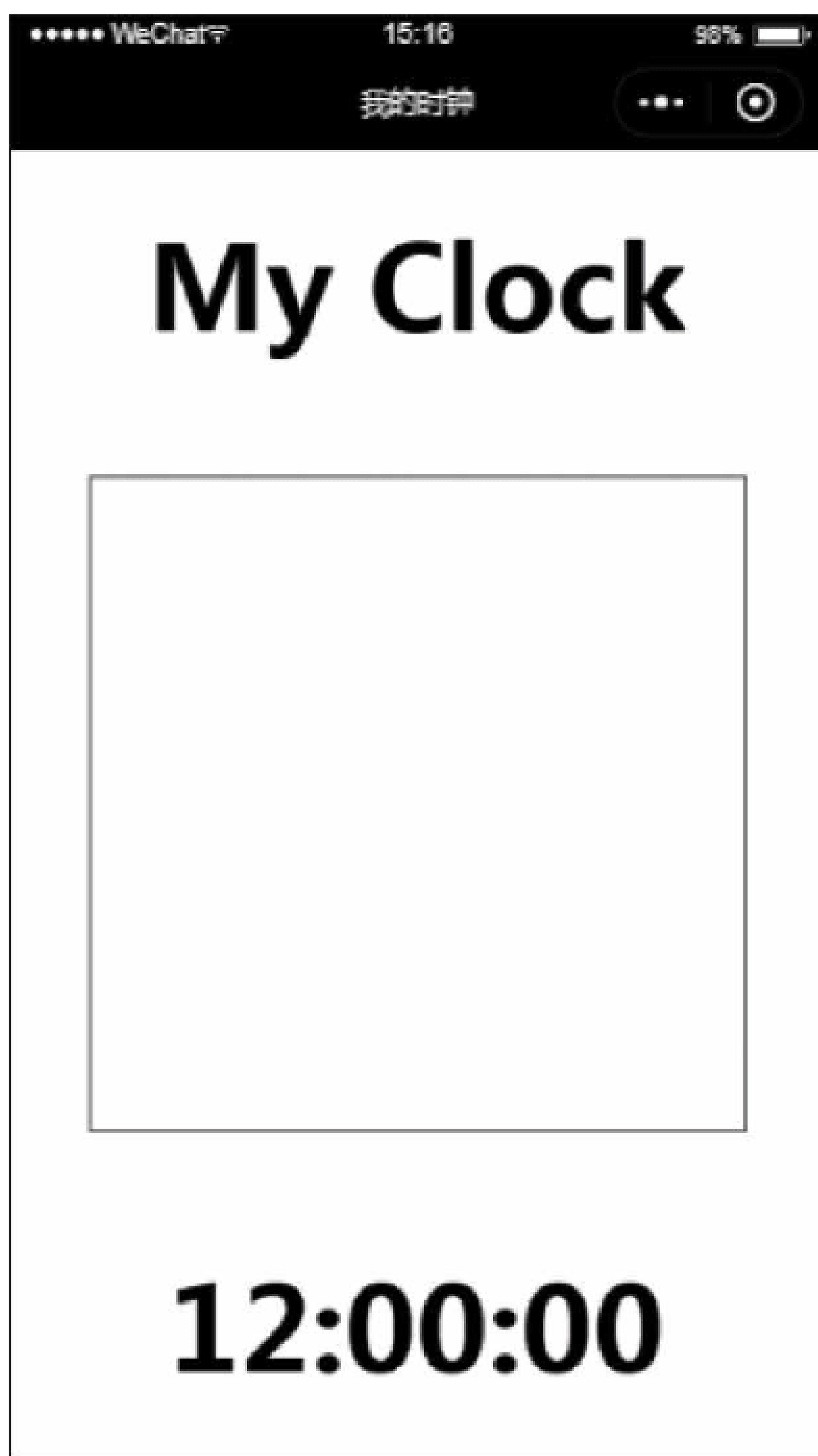


图 11-8　数字电子时钟区域预览效果

此时页面设计就全部完成了，接下来需要进行逻辑实现。



11.4 逻辑实现



11.4.1 创建画布上下文



视频讲解

首先需要根据画布组件的 canvas-id 属性在 JS 文件的生命周期函数 onLoad 中创建画布上下文(CanvasContext),然后才可以进行绘制工作。

JS(pages/index/index.js)代码片段修改如下:

```
1.  Page({
2.    /**
3.     * 生命周期函数--监听页面加载
4.     */
5.    onLoad: function(options) {
6.    //创建画布上下文
7.      this.ctx = wx.createCanvasContext('clockCanvas')
8.    },
9.  })
```

其中引号里面的内容就是 WXML 页面中画布组件的 canvas-id 属性值。

这里可以简单测试一下画布上下文是否已经生效,在刚才的代码下方追加两句临时代码,绘制一个矩形。

JS(pages/index/index.js)代码片段临时修改如下:

```
1.  Page({
2.    /**
3.     * 生命周期函数--监听页面加载
4.     */
5.    onLoad: function(options) {
6.      //创建画布上下文
7.      this.ctx = wx.createCanvasContext('clockCanvas')
8.      //设置画笔的填充颜色为红色
9.      this.ctx.fillStyle = 'red'
10.     //设置矩形为左上角在画布的(0,0)坐标点,长和宽
        //均为 300 像素
11.     this.ctx.fillRect(0,0,300,300)
12.     //在画布上绘制出来
13.     this.ctx.draw()
14.   },
15. })
```



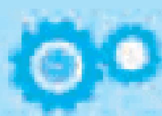
图 11-9 画布上下文的临时使用效果

当前效果如图 11-9 所示。

由图可见,此时已经可以利用画布上下文进行一些简单的图形绘制工作了。在测试完成后可以删除或注释掉绘制红色填充矩形的代码,等待正式开发电子时钟。

11.4.2 绘制时钟刻度

在 JS 文件中创建自定义函数 drawClock 用于进行电子时钟的绘制,并在 onLoad 函数中进行调用。JS(pages/



index/index.js)代码片段修改如下：

```
1.  Page({
2.    /**
3.     * 绘制时钟
4.     */
5.    drawClock: function() {
6.      /*1.准备工作*/
7.      //定义时钟的宽和高(默认单位: px)
8.      let width = 300, height = 300
9.      //获取画布上下文
10.     var ctx = this.ctx
11.     //设置画布中心为参照点
12.     ctx.translate(width / 2, height / 2)
13.     //将画布逆时针旋转 90°
14.     ctx.rotate( - Math.PI / 2)
15.   },
16.   /**
17.    * 生命周期函数--监听页面加载
18.    */
19.   onLoad: function(options) {
20.     //创建画布上下文
21.     this.ctx = wx.createCanvasContext('clockCanvas')
22.     //绘制时钟
23.     this.drawClock()
24.   },
25. })
```

上述代码首先规定了电子时钟的画面尺寸为宽、高均为 300 像素，并获取画布上下文；然后使用 translate()方法将参照点移动到了画布的中心坐标(150,150)，表示未来将以画布中心坐标为参照点进行变形、位移、旋转等设置。由于画布默认是从水平向右的方向进行圆弧的绘制，所以使用 rotate()方法先把画布整体逆时针旋转 90°，表示将从水平向上的方向开始进行绘制，这样方便进行弧度计算。

1 绘制小时刻度

首先绘制 12 个小时对应的刻度，相邻两个刻度之间的间隔为 30°，换算成弧度单位为 $\pi/6$。因此可以使用 for 循环语句循环 12 次，从之前设置的水平向上的方向开始绘制第一条刻度，然后再使用 rotate()方法顺时针旋转 30°绘制下一条刻度，直到全部完成为止。



视频讲解

JS(pages/index/index.js)代码片段修改如下：

```
1.  Page({
2.    /**
3.     * 绘制时钟
4.     */
5.    drawClock: function() {
6.      /*1.准备工作*/
7.      代码略
8.
9.      /*2.绘制时钟刻度*/
10.     /*2-1.绘制小时刻度*/
11.     //设置线条粗细
12.     ctx.lineWidth = 6
```

```
13.     //设置线条末端样式
14.     ctx.lineCap = 'round'
15.     //绘制 12 个小时刻度
16.     for (let i = 0; i < 12; i++) {
17.       //开始路径
18.       ctx.beginPath()
19.       //从(100,0)绘制到(120,0)
20.       ctx.moveTo(100, 0)
21.       ctx.lineTo(120, 0)
22.       //描边路径
23.       ctx.stroke()
24.       //顺时针旋转 30°
25.       ctx.rotate(Math.PI / 6)
26.     }
27.
28.     /* 绘制所有内容 */
29.     ctx.draw()
30.   },
31. })
```

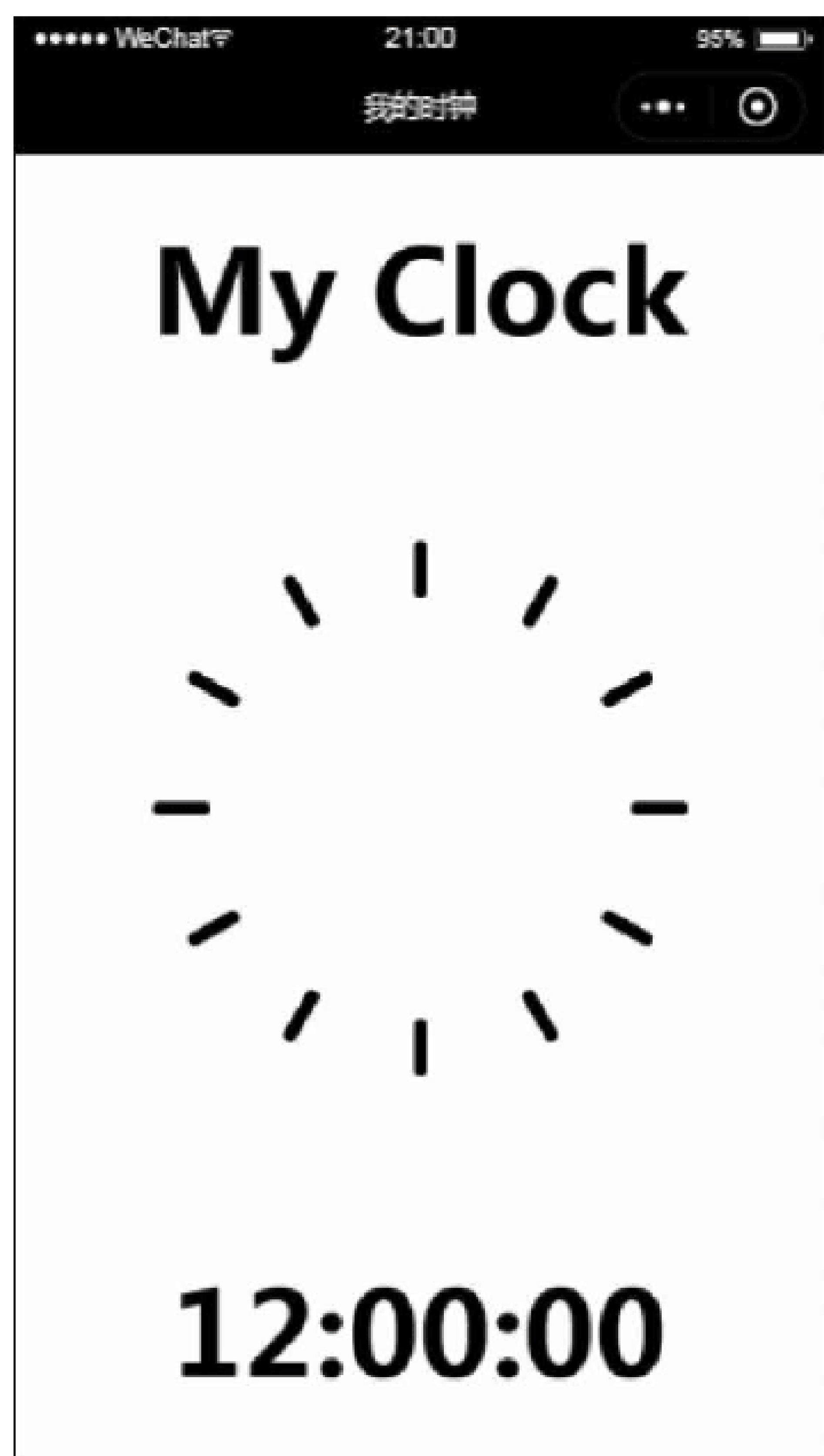


图 11-10 绘制小时刻度的效果

当前效果如图 11-10 所示。

由图可见，此时已经完成 12 个小时刻度的绘制了。

2 绘制分钟刻度

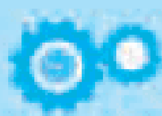


视频讲解

然后绘制 60 分钟对应的刻度，相邻两个刻度之间的间隔为 6°，换算成弧度单位为 π/30。因此可以使用 for 循环语句循环 60 次，从之前设置的水平向上的方向开始绘制第一条刻度，然后再使用 rotate()方法顺时针旋转 6°绘制下一条刻度，直到全部完成为止。

JS(pages/index/index.js)代码片段修改如下：

```
1.  Page({
2.    /**
3.     * 绘制时钟
4.     */
5.    drawClock: function() {
6.      /* 1.准备工作 */
7.      代码略
8.
9.      /* 2.绘制时钟刻度 */
10.     /* 2-1.绘制小时刻度 */
11.     代码略
12.
13.     /* 2-2.绘制分钟刻度 */
14.     //设置线条粗细
15.     ctx.lineWidth = 5
16.     //设置线条末端样式
17.     ctx.lineCap = 'round'
18.     //绘制 60 个分钟刻度
19.     for (let i = 0; i < 60; i++) {
20.       //开始绘制路径
21.       ctx.beginPath()
22.       //从(118,0)绘制到(120,0)
23.       ctx.moveTo(118, 0)
```



```
24.         ctx.lineTo(120, 0)
25.         //描边路径
26.         ctx.stroke()
27.         //顺时针旋转 6°
28.         ctx.rotate(Math.PI / 30)
29.       }
30.
31.       /* 绘制所有内容 */
32.       ctx.draw()
33.     },
34.  })
```

当前效果如图 11-11 所示。

由图可见，此时已经完成时钟全部刻度的绘制了，接下来将绘制时钟指针。

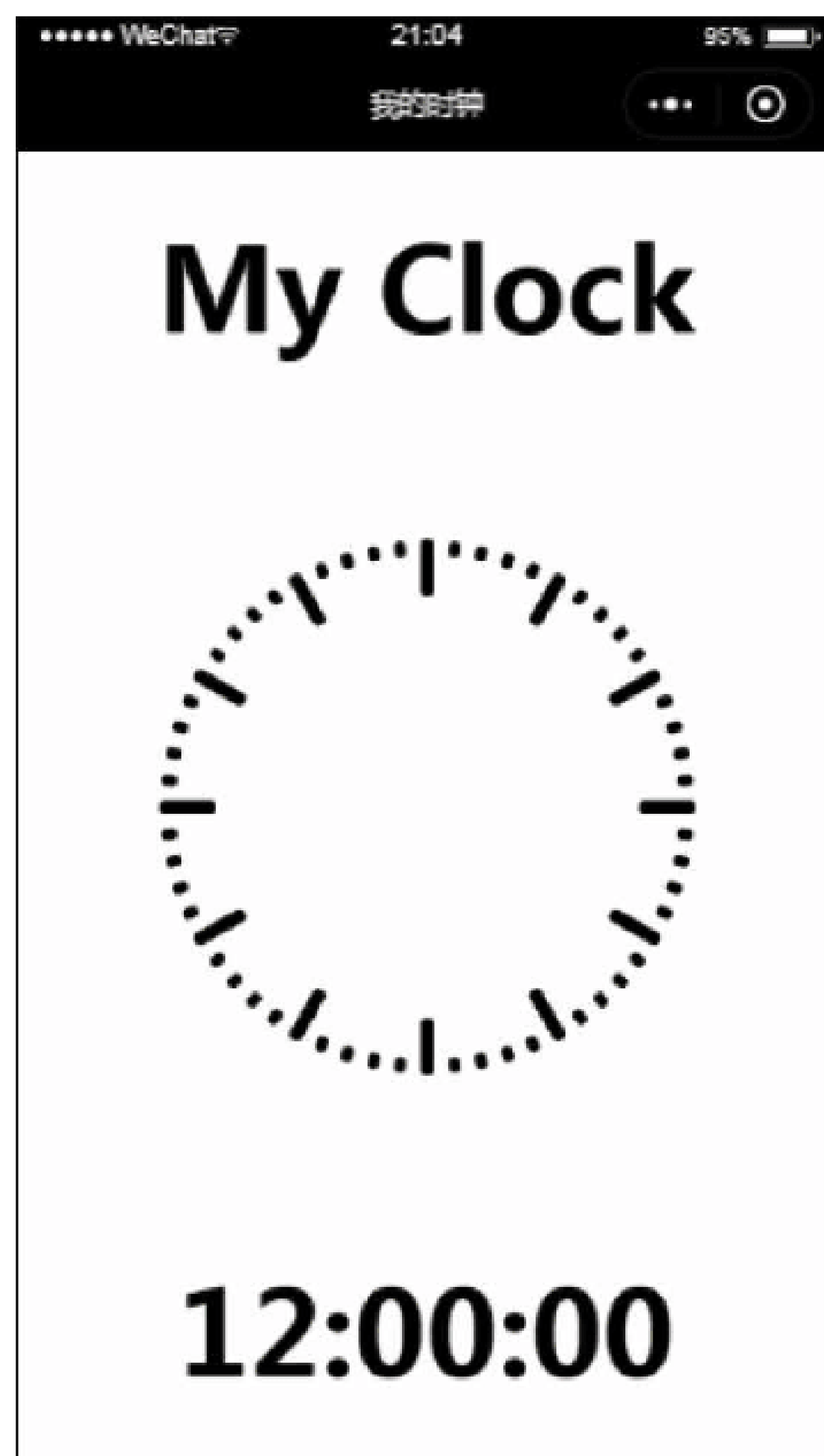


图 11-11　绘制分钟刻度的效果

11.4.3　绘制时钟指针

在绘制时钟指针之前需要获得当前的时间信息。在 JS 文件中创建自定义函数 getTime 用于获取当前时间，并在 drawClock 函数中进行调用。

JS(pages/index/index.js)代码片段修改如下：

```
1.   Page({
2.     /**
3.      * 获取当前时间
4.      */
5.     getTime: function() {
6.       let now = new Date()              //获取当前时间日期对象
7.       let time = []                     //声明一个空数组用于存放时、分、秒
8.       time[0] = now.getHours()          //获得小时
9.       time[1] = now.getMinutes()        //获得分钟
10.      time[2] = now.getSeconds()        //获得秒钟
11.
12.      //将 24 小时制转换为 12 小时制
13.      if (time[0] > 12)
14.        time[0] -= 12
15.
16.      //返回时分秒数组
17.      return time
18.    },
19.    /**
20.     * 绘制时钟
21.     */
22.    drawClock: function() {
23.      /* 1.准备工作 */
24.      代码略
25.
26.      /* 2.绘制时钟刻度 */
27.      代码略
28.
29.      /* 3.获取当前时间 */
30.      let time = this.getTime()         //获取当前时间
```

```
31.     let h = time[0]                    //获取小时
32.     let m = time[1]                    //获取分钟
33.     let s = time[2]                    //获取秒钟
34.
35.     /*绘制所有内容*/
36.     ctx.draw()
37.   },
38. })
```

开发者可以自行使用 console.log(time)语句测试控制台是否可以获取当前时间。



视频讲解

1 绘制时针

首先绘制时针，以12点方向的刻度为参照，当前时针需要顺时针旋转的角度为：

时针的角度=360°/12 * h+360°/12/60 * m+360°/12/60/60 * s

换算成弧度单位如下：

时针的弧度=π/6 * h+π/360 * m+π/21600 * s

因此先根据公式计算时针需要旋转的弧度，然后进行绘制。

JS(pages/index/index.js)代码片段修改如下：

```
1.  Page({
2.    /**
3.     * 绘制时钟
4.     */
5.    drawClock: function() {
6.      /*1.准备工作*/
7.      代码略
8.
9.      /*2.绘制时钟刻度*/
10.     代码略
11.
12.     /*3.获取当前时间*/
13.     代码略
14.
15.     /*4.绘制时钟指针*/
16.     /*4-1.绘制时针*/
17.     //保存当前的绘图状态
18.     ctx.save()
19.     //旋转角度
20.     ctx.rotate(h * Math.PI / 6 + m * Math.PI / 360 + s * Math.PI / 21600)
21.     //设置线条粗细
22.     ctx.lineWidth = 12
23.     //开始绘制路径
24.     ctx.beginPath()
25.     //从(-20,0)绘制到(80,0)
26.     ctx.moveTo(-20, 0)
27.     ctx.lineTo(80, 0)
28.     //描边路径
29.     ctx.stroke()
30.     //恢复之前保存的绘图样式
31.     ctx.restore()
32.
```

```
33.        /*绘制所有内容*/
34.        ctx.draw()
35.    },
36. })
```

当前效果如图 11-12 所示。



图 11-12　绘制时针的效果

2 绘制分针



视频讲解

然后绘制分针，以 12 点方向的刻度为参照，当前分针需要顺时针旋转的角度为：

$$分针的角度=360°/60*m+360°/60/60*s$$

换算成弧度单位如下：

$$分针的弧度=\pi/30*m+\pi/1800*s$$

因此先根据公式计算分针需要旋转的弧度，然后进行绘制。

JS(pages/index/index.js)代码片段修改如下：

```
1.  Page({
2.    /**
3.     * 绘制时钟
4.     */
5.    drawClock: function() {
6.      /*1.准备工作*/
7.      代码略
8.
9.      /*2.绘制时钟刻度*/
10.     代码略
11.
12.     /*3.获取当前时间*/
13.     代码略
```

```
14.
15.     /*4.绘制时钟指针*/
16.     /*4-1.绘制时针*/
17.     代码略
18.     /*4-2.绘制分针*/
19.     //保存当前的绘图状态
20.     ctx.save()
21.     //旋转角度
22.     ctx.rotate(m * Math.PI / 30 + s * Math.PI / 1800)
23.     //设置线条粗细
24.     ctx.lineWidth = 8
25.     //开始绘制路径
26.     ctx.beginPath()
27.     //从(-20,0)绘制到(112,0)
28.     ctx.moveTo(-20, 0)
29.     ctx.lineTo(112, 0)
30.     //描边路径
31.     ctx.stroke()
32.     //恢复之前保存的绘图样式
33.     ctx.restore()
34.
35.     /*绘制所有内容*/
36.     ctx.draw()
37.   },
38. })
```



图 11-13 绘制分针的效果

当前效果如图 11-13 所示。

3 绘制秒针

最后绘制秒针，以 12 点方向的刻度为参照，当前秒针需要顺时针旋转的角度为：



视频讲解

$$秒针的角度=360°/60*s$$

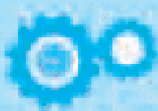
换算成弧度单位如下：

$$秒针的弧度=\pi/30*s$$

因此先根据公式计算秒针需要旋转的弧度，然后进行绘制。

JS(pages/index/index.js)代码片段修改如下：

```
1.  Page({
2.    /**
3.     * 绘制时钟
4.     */
5.    drawClock: function() {
6.      /*1.准备工作*/
7.      代码略
8.
9.      /*2.绘制时钟刻度*/
10.     代码略
11.
12.     /*3.获取当前时间*/
13.     代码略
14.
15.     /*4.绘制时钟指针*/
16.     /*4-1.绘制时针*/
17.     代码略
```



```
18.        /* 4 - 2.绘制分针 */
19.        代码略
20.        /* 4 - 3.绘制秒针 */
21.        //保存当前的绘图状态
22.        ctx.save()
23.        //旋转角度
24.        ctx.rotate(s * Math.PI / 30)
25.        //设置画笔描边颜色为红色
26.        ctx.strokeStyle = 'red'
27.        //设置线条粗细
28.        ctx.lineWidth = 6
29.        //开始绘制路径
30.        ctx.beginPath()
31.        //从(-30,0)绘制到(120,0)
32.        ctx.moveTo(-30, 0)
33.        ctx.lineTo(120, 0)
34.        //描边路径
35.        ctx.stroke()
36.
37.        //设置填充颜色为红色
38.        ctx.fillStyle = 'red'
39.        //开始绘制路径
40.        ctx.beginPath()
41.        //绘制圆弧
42.        ctx.arc(0, 0, 10, 0, Math.PI * 2, true)
43.        //填充圆弧
44.        ctx.fill()
45.        //恢复之前保存的绘图样式
46.        ctx.restore()
47.
48.        /* 绘制所有内容 */
49.        ctx.draw()
50.    },
51. })
```



图 11-14　绘制秒针的效果

当前效果如图 11-14 所示。

由图可见，此时已经完成时钟全部指针的绘制了，接下来将更新数字电子时钟的信息。

11.4.4　显示数字电子时钟


视频讲解

WXML(pages/index/index.wxml)代码片段修改如下：

```
1.  <view class = 'container'>
2.    <text> My Clock </text>
3.    <canvas canvas-id = 'clockCanvas'></canvas>
4.    <text>{{h}}:{{m}}:{{s}}</text>
5.  </view>
```

JS(pages/index/index.js)代码片段修改如下：

```
1.  Page({
2.    /**
3.     * 绘制时钟
4.     */
5.    drawClock: function() {
```

```
6.        /*1.准备工作*/
7.        代码略
8.
9.        /*2.绘制时钟刻度*/
10.       代码略
11.
12.       /*3.获取当前时间*/
13.       代码略
14.
15.       /*4.绘制时钟指针*/
16.       代码略
17.
18.       /*绘制所有内容*/
19.       ctx.draw()
20.
21.       /*更新页面显示时间*/
22.       this.setData({
23.         h: h > 9 ? h : '0' + h,
24.         m: m > 9 ? m : '0' + m,
25.         s: s > 9 ? s : '0' + s
26.       })
27.     },
28. })
```

当前效果如图 11-15 所示。

由图可见，已经成功更新数字电子时钟的信息。



图 11-15 更新数字电子时钟的效果

11.4.5 每秒实时更新

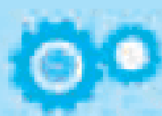


视频讲解

用户可以在 JS 文件的 onLoad 函数中使用 setInterval 函数设置每秒重新刷新画面，从而实现动画效果，并在 onUnload 函数中清除计时器。

JS(pages/index/index.js)代码片段修改如下：

```
1.  Page({
2.    /**
3.     * 生命周期函数--监听页面加载
4.     */
5.    onLoad: function(options) {
6.      //创建画布上下文
7.      this.ctx = wx.createCanvasContext('clockCanvas')
8.      //绘制时钟
9.      this.drawClock()
10.
11.     var that = this
12.     //每秒更新绘制
13.     this.interval = setInterval(function() {
14.       that.drawClock()
15.     }, 1000)
16.   },
17.   /**
18.    * 生命周期函数--监听页面卸载
19.    */
20.   onUnload: function() {
21.     clearInterval(this.interval)
```

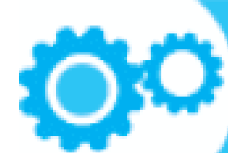
```
22.    },
23. })
```

当前效果如图 11-16 所示。



图 11-16　时钟的动态效果

此时手绘时钟小程序就全部完成了，开发者可以自行更换时钟的颜色和尺寸效果。



11.5　完整代码展示



app.json 文件的完整代码如下：

```
1.  {
2.    "pages": [
3.      "pages/index/index"
4.    ],
5.    "window": {
6.      "navigationBarTitleText": "我的时钟"
7.    }
8.  }
```

WXML 文件(pages/index/index.wxml)的完整代码如下：

```
1.  <view class = 'container'>
2.    <text> My Clock </text>
3.    <canvas canvas - id = 'clockCanvas'></canvas>
4.    <text>{{h}}:{{m}}:{{s}}</text>
5.  </view>
```

WXSS 文件(pages/index/index.wxss)的完整代码如下：

```
1.  /* 整体容器样式 */
2.  .container{
```

```
3.     height: 100vh;                      /* 高度 */
4.     display: flex;                      /* flex 布局模型 */
5.     flex-direction: column;             /* 垂直布局 */
6.     align-items: center;                /* 水平方向居中 */
7.     justify-content: space-around;      /* 调整内容间隙 */
8.   }
9.   /* 文本样式 */
10.  text{
11.    font-size: 40pt;                    /* 字号大小 */
12.    font-weight: bold;                  /* 字体加粗 */
13.  }
14.  /* 画布样式 */
15.  canvas{
16.    width: 600rpx;                      /* 宽度 */
17.    height: 600rpx;                     /* 高度 */
18.  }
```

JS 文件(pages/index/index.js)的完整代码如下：

```
1.   Page({
2.     /**
3.       * 获取当前时间
4.       */
5.     getTime: function() {
6.       let now = new Date()              //获取当前时间日期对象
7.       let time = []                     //声明一个空数组用于存放时、分、秒
8.       time[0] = now.getHours()          //获得小时
9.       time[1] = now.getMinutes()        //获得分钟
10.      time[2] = now.getSeconds()        //获得秒钟
11.
12.      //将 24 小时制转换为 12 小时制
13.      if (time[0] > 12)
14.        time[0] -= 12
15.
16.      //返回时分秒数组
17.      return time
18.    },
19.    /**
20.      * 绘制时钟
21.      */
22.    drawClock: function() {
23.      /* 1.准备工作 */
24.      //定义时钟的宽和高(默认单位: px)
25.      let width = 300,
26.        height = 300
27.      //获取画布上下文
28.      var ctx = this.ctx
29.      //设置画布中心为参照点
30.      ctx.translate(width / 2, height / 2)
31.      //将画布逆时针旋转 90°
32.      ctx.rotate(-Math.PI / 2)
33.
34.      /* 2.绘制时钟刻度 */
35.      /* 2-1.绘制小时刻度 */
36.      //设置线条粗细
37.      ctx.lineWidth = 6
```

```
38.    //设置线条末端样式
39.    ctx.lineCap = 'round'
40.    //绘制 12 个小时刻度
41.    for (let i = 0; i < 12; i++) {
42.      //开始路径
43.      ctx.beginPath()
44.      //从(100,0)绘制到(120,0)
45.      ctx.moveTo(100, 0)
46.      ctx.lineTo(120, 0)
47.      //描边路径
48.      ctx.stroke()
49.      //顺时针旋转 30°
50.      ctx.rotate(Math.PI / 6)
51.    }
52.
53.    /* 2 - 2.绘制分钟刻度 */
54.    //设置线条粗细
55.    ctx.lineWidth = 5
56.    //设置线条末端样式
57.    ctx.lineCap = 'round'
58.    //绘制 60 个分钟刻度
59.    for (let i = 0; i < 60; i++) {
60.      //开始绘制路径
61.      ctx.beginPath()
62.      //从(118,0)绘制到(120,0)
63.      ctx.moveTo(118, 0)
64.      ctx.lineTo(120, 0)
65.      //描边路径
66.      ctx.stroke()
67.      //顺时针旋转 6°
68.      ctx.rotate(Math.PI / 30)
69.    }
70.
71.    /* 3.获取当前时间 */
72.    let time = this.getTime()          //获取当前时间
73.    let h = time[0]                    //获取小时
74.    let m = time[1]                    //获取分钟
75.    let s = time[2]                    //获取秒钟
76.
77.    /* 4.绘制时钟指针 */
78.    /* 4 - 1.绘制时针 */
79.    //保存当前的绘图状态
80.    ctx.save()
81.    //旋转角度
82.    ctx.rotate(h * Math.PI / 6 + m * Math.PI / 360 + s * Math.PI / 21600)
83.    //设置线条粗细
84.    ctx.lineWidth = 12
85.    //开始绘制路径
86.    ctx.beginPath()
87.    //从(-20,0)绘制到(80,0)
88.    ctx.moveTo(-20, 0)
89.    ctx.lineTo(80, 0)
90.    //描边路径
91.    ctx.stroke()
92.    //恢复之前保存的绘图样式
```

```
93.     ctx.restore()
94.
95.     /* 4 - 2.绘制分针 */
96.     //保存当前的绘图状态
97.     ctx.save()
98.     //旋转角度
99.     ctx.rotate(m * Math.PI / 30 + s * Math.PI / 1800)
100.    //设置线条粗细
101.    ctx.lineWidth = 8
102.    //开始绘制路径
103.    ctx.beginPath()
104.    //从(-20,0)绘制到(112,0)
105.    ctx.moveTo(-20, 0)
106.    ctx.lineTo(112, 0)
107.    //描边路径
108.    ctx.stroke()
109.    //恢复之前保存的绘图样式
110.    ctx.restore()
111.
112.    /* 4 - 3.绘制秒针 */
113.    //保存当前的绘图状态
114.    ctx.save()
115.    //旋转角度
116.    ctx.rotate(s * Math.PI / 30)
117.    //设置画笔描边颜色为红色
118.    ctx.strokeStyle = 'red'
119.    //设置线条粗细
120.    ctx.lineWidth = 6
121.    //开始绘制路径
122.    ctx.beginPath()
123.    //从(-30,0)绘制到(120,0)
124.    ctx.moveTo(-30, 0)
125.    ctx.lineTo(120, 0)
126.    //描边路径
127.    ctx.stroke()
128.
129.    //设置填充颜色为红色
130.    ctx.fillStyle = 'red'
131.    //开始绘制路径
132.    ctx.beginPath()
133.    //绘制圆弧
134.    ctx.arc(0, 0, 10, 0, Math.PI * 2, true)
135.    //填充圆弧
136.    ctx.fill()
137.    //恢复之前保存的绘图样式
138.    ctx.restore()
139.
140.    /* 绘制所有内容 */
141.    ctx.draw()
142.
143.    /* 更新页面显示时间 */
144.    this.setData({
145.      h: h > 9 ? h : '0' + h,
146.      m: m > 9 ? m : '0' + m,
147.      s: s > 9 ? s : '0' + s
```

```
148.     })
149.   },
150.   /**
151.    * 生命周期函数--监听页面加载
152.    */
153.   onLoad: function(options) {
154.     //创建画布上下文
155.     this.ctx = wx.createCanvasContext('clockCanvas')
156.     //绘制时钟
157.     this.drawClock()
158.
159.     var that = this
160.     //每秒更新绘制
161.     this.interval = setInterval(function() {
162.       that.drawClock()
163.     }, 1000)
164.   },
165.   /**
166.    * 生命周期函数--监听页面卸载
167.    */
168.   onUnload: function() {
169.     clearInterval(this.interval)
170.   },
171. })
```

游戏篇



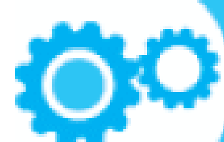
第12章 Chapter 12

小程序游戏·拼图游戏

在学习了<canvas>(画布)组件和小程序界面API中绘图的相关用法以后,读者不妨尝试制作简易的拼图小游戏。

本章学习目标

- 综合应用所学知识创建完整的拼图游戏项目;
- 熟练掌握<canvas>组件和绘图API。



12.1 需求分析





视频讲解

本项目一共需要两个页面,即首页和游戏页面,其中,首页用于呈现关卡菜单,点击对应难度的关卡后进入游戏画面。

12.1.1 首页功能需求

首页功能需求如下:

(1) 首页需要包含标题和关卡列表。

(2) 关卡至少要有6个关卡选项,每个关卡显示预览图片和第几关。

(3) 点击关卡列表可以打开对应的游戏画面。

12.1.2 游戏页功能需求

游戏页功能需求如下。

(1) 游戏页面需要显示游戏提示图、游戏画面和"重新开始"按钮。

(2) 每关游戏提示图显示对应的图片预览。

(3) 游戏画面随机将原图打乱为3×3的小方块,并且可移动被点击的方块。

(4) 点击"重新开始"按钮可以重新随机打乱小方块并开始游戏。



12.2 项目创建





视频讲解

本项目创建选择空白文件夹jigsawGame,效果如图12-1所示。

单击"新建"按钮完成项目创建,然后准备手动修改页面配置文件。

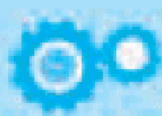
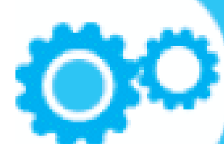




图 12-1　小程序项目填写效果示意图



12.3　页面配置

12.3.1　创建页面文件





视频讲解

项目创建完毕后，在根目录中会生成文件夹 pages 用于存放页面文件。一般来说首页默认命名为 index，表示小程序运行的第一个页面；其他页面名称可以自定义。本项目有两个页面文件，需要创建 index（首页页面）和 game（游戏页面）。

具体操作如下：

（1）将 app.json 文件内 pages 属性中的"pages/logs/logs"改成"pages/game/game"。

（2）按快捷键 Ctrl+S 保存修改后会在 pages 文件夹下自动生成 game 目录。

12.3.2　删除和修改文件

具体操作如下：

（1）删除 utils 文件夹及其内部所有内容。

（2）删除 pages 文件夹下的 logs 目录及其内部所有内容。

（3）删除 index.wxml 和 index.wxss 中的全部代码。

（4）删除 index.js 中的全部代码，并且输入关键词 page 找到第二个选项按回车键让其自动补全函数（如图 12-2 所示）。

（5）删除 app.wxss 中的全部代码。

（6）删除 app.js 中的全部代码，并且输入关键词 app 找到第一个选项按回车键让其自动补全函数（如图 12-3 所示）。

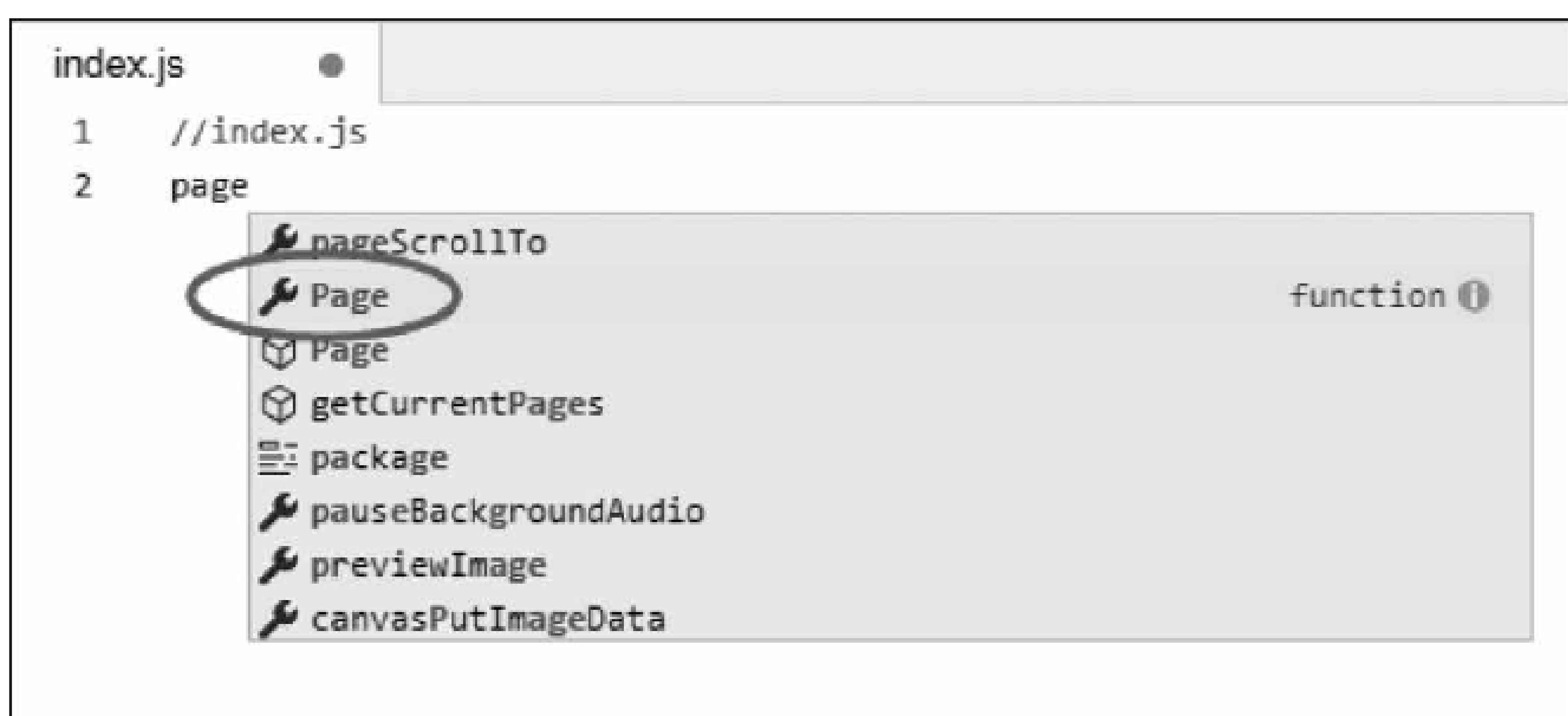


图 12-2 输入关键词创建 Page 函数

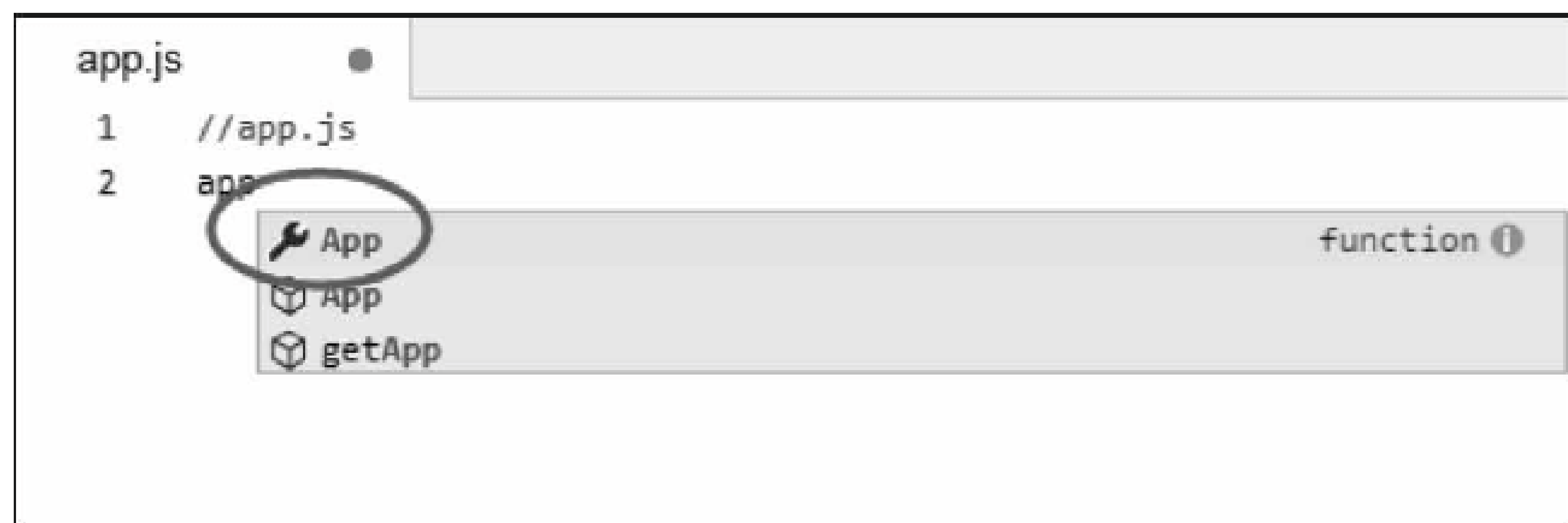


图 12-3 输入关键词创建 App 函数

12.3.3 创建其他文件

接下来创建其他自定义文件，本项目还需要一个 images 文件夹用于存放关卡图片。单击目录结构左上角的＋号创建文件夹并命名为 images，该文件夹名称可以由开发者自定义。

图片素材展示如图 12-4 所示。



(a) pic01.jpg



(b) pic02.jpg



(c) pic03.jpg



(d) pic04.jpg



(e) pic05.jpg



(f) pic06.jpg

图 12-4 图片素材展示

右击目录结构中的 images 文件夹，选择“硬盘打开”，将图片复制、粘贴进去。

全部完成后的目录结构如图 12-5 所示。

此时文件配置就全部完成，12.4 节将正式进行页面布局和样式设计。

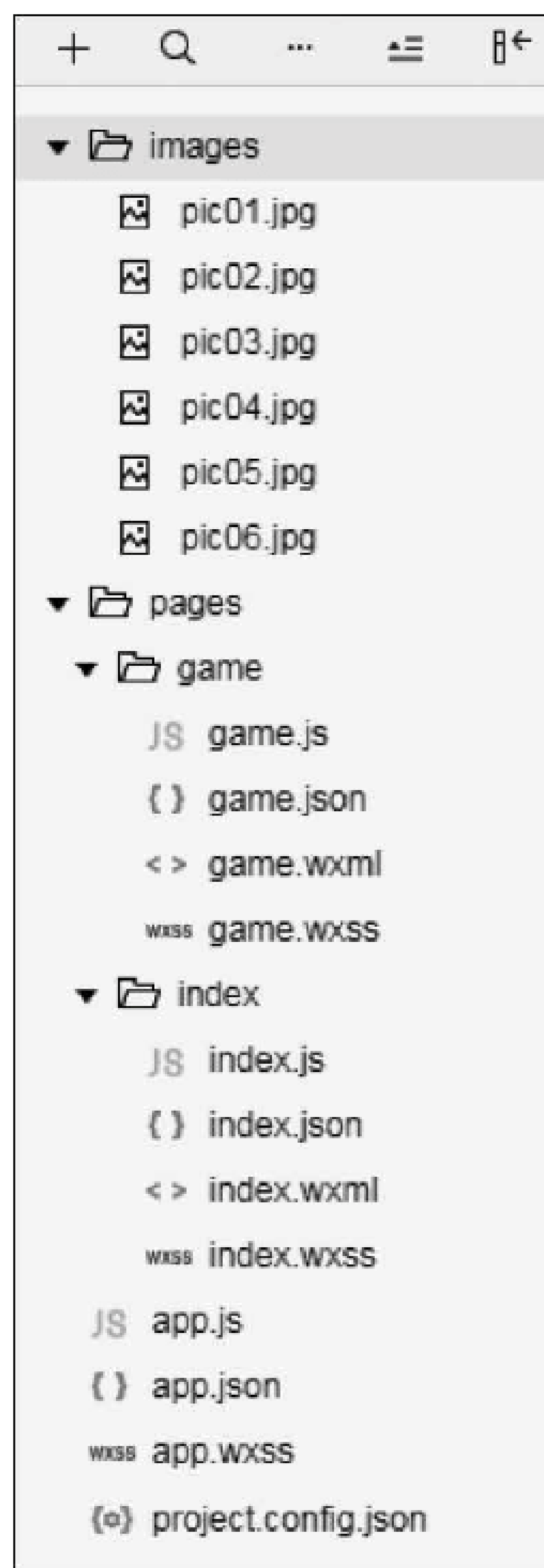
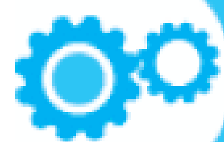


图 12-5　页面文件创建完成



12.4　视图设计



12.4.1　导航栏设计



视频讲解

小程序默认导航栏是黑底白字的效果，可以通过在 app. json 中对 window 属性进行重新配置来自定义导航栏效果。更改后的 app. json 文件代码如下：

```
1.  {
2.    "pages": [代码略],
3.    "window": {
4.      "navigationBarBackgroundColor": "#E64340",
5.      "navigationBarTitleText": "拼图游戏"
6.    }
7.  }
```

上述代码可以更改导航栏背景色为珊瑚红色，字体为白色，效果如图 12-6 所示。



图 12-6　自定义导航栏效果

12.4.2　页面设计



视频讲解

1 公共样式设计

首先在 app. wxss 中设置页面容器和顶端标题的公共

样式，代码如下：

```
1.  /* 页面容器样式 */
2.  .container {
3.    height: 100vh;
4.    color: #E64340;
5.    font-weight: bold;
6.    display: flex;
7.    flex-direction: column;
8.    align-items: center;
9.    justify-content: space-evenly;
10. }
11. /* 顶端标题样式 */
12. .title {
13.   font-size: 18pt;
14. }
```

2 首页设计



视频讲解

首页主要包含两部分内容，即标题和关卡列表，页面设计如图 12-7 所示。

计划使用如下组件。

- 顶端标题：<view>容器；
- 关卡列表：<view>容器，内部使用数组循环。

WXML(pages/index/index.wxml)代码如下：

```
1.  <view class='container'>
2.    <!-- 标题 -->
3.    <view class='title'>游戏选关</view>
4.
5.    <!-- 关卡列表 -->
6.    <view class='levelBox'>
7.      <view class='box'>
8.        <image src='/images/pic01.jpg'></image>
9.        <text>第 1 关</text>
10.     </view>
11.   </view>
12.
13. </view>
```

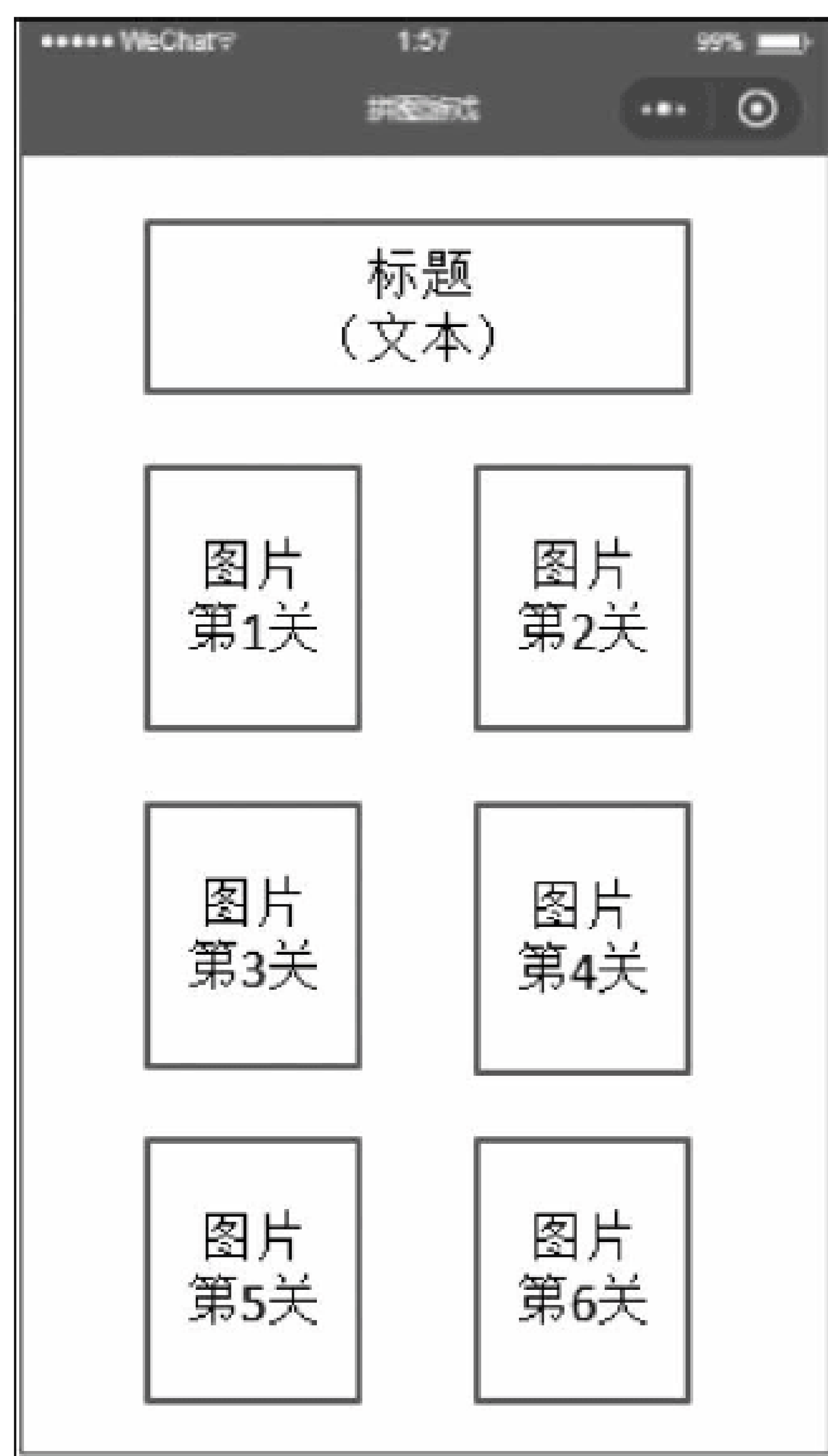
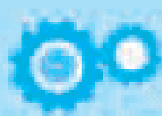


图 12-7 首页设计图

相关 WXSS(pages/index/index.wxss)代码片段如下：

```
1.  /* 关卡列表区域 */
2.  .levelBox{
3.    width: 100%;
4.  }
5.  /* 单个关卡区域 */
6.  .box{
7.    width: 50%;
8.    float: left;
9.    margin: 25rpx 0;
10.   display: flex;
11.   flex-direction: column;
12.   align-items: center;
13. }
14. /* 选关图片 */
15. image{
16.   width: 260rpx;
```



```
17.    height: 260rpx;
18. }
```

当前效果如图 12-8 所示。

由图可见,此时可以显示标题和一个临时关卡。由于尚未获得关卡数据,所以暂时无法显示完整的关卡列表,仅供作为样式参考。



视频讲解

3 游戏页面设计

游戏页面需要用户点击首页的关卡列表,然后在新窗口中打开该页面。游戏页面包括游戏提示图、游戏画面和"重新开始"按钮,页面设计如图 12-9 所示。



图 12-8 首页效果图

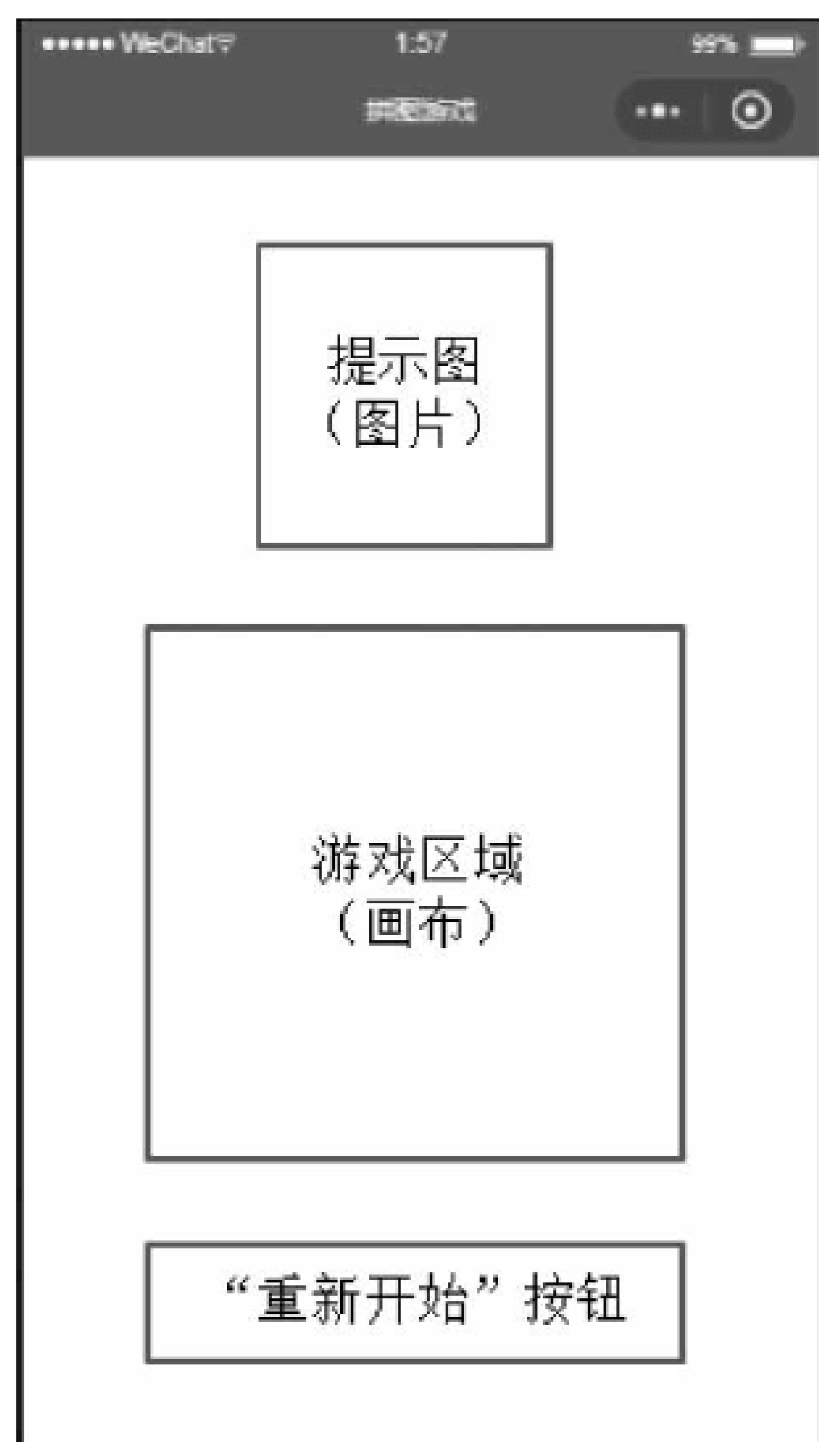


图 12-9 游戏页设计图

由于暂时没有做点击跳转的逻辑设计,所以可以在开发工具顶端选择"普通编译"下的"添加编译模式",并携带临时测试参数 level=pic01.jpg,如图 12-10 所示。

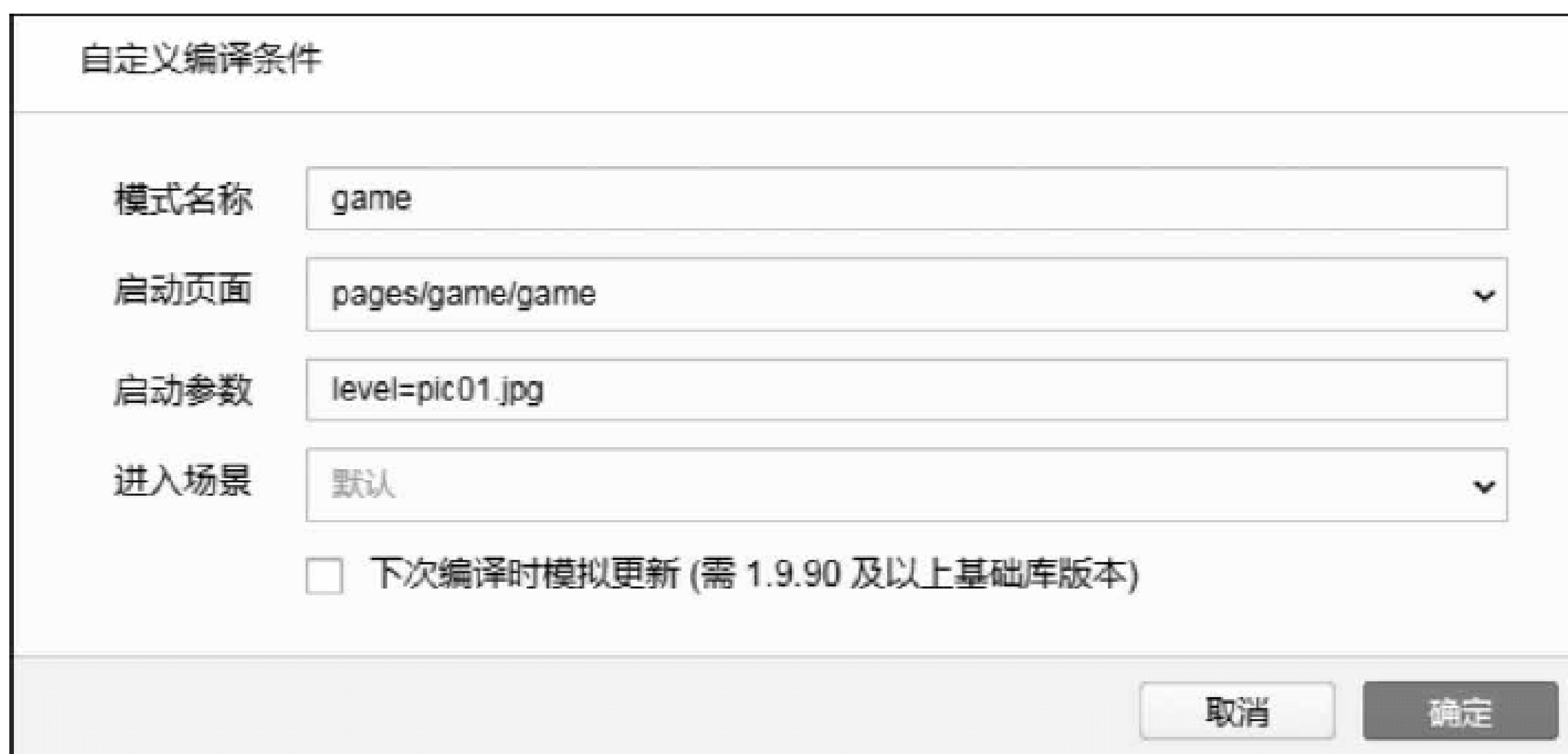


图 12-10 添加 game 页面的编译模式

此时预览就可以直接显示 game 页面了,设计完毕后再改回"普通编译"模式即可重新显示首页。

计划使用如下组件。

- <view>：整体容器和顶端标题；
- <image>：提示图；
- <canvas>：游戏画布；
- <button>："重新开始"按钮。

WXML(pages/game/game.wxml)代码如下：

```
1.  <view class = 'container'>
2.    <!-- 提示图区域 -->
3.    <view class = 'title'>提示图</view>
4.    <image src = '/images/pic01.jpg'></image>
5.
6.    <!-- 游戏画布 -->
7.    <canvas canvas - id = 'myCanvas'></canvas>
8.
9.    <!-- "重新开始"按钮 -->
10.   <button type = 'warn'>重新开始</button>
11. </view>
```

WXSS(pages/game/game.wxss)代码如下：

```
1.  /* 提示图样式 */
2.  image {
3.    width: 250rpx;
4.    height: 250rpx;
5.  }
6.
7.  /* 画布样式 */
8.  canvas {
9.    border: 1rpx solid;
10.   width: 300px;
11.   height: 300px;
12. }
```

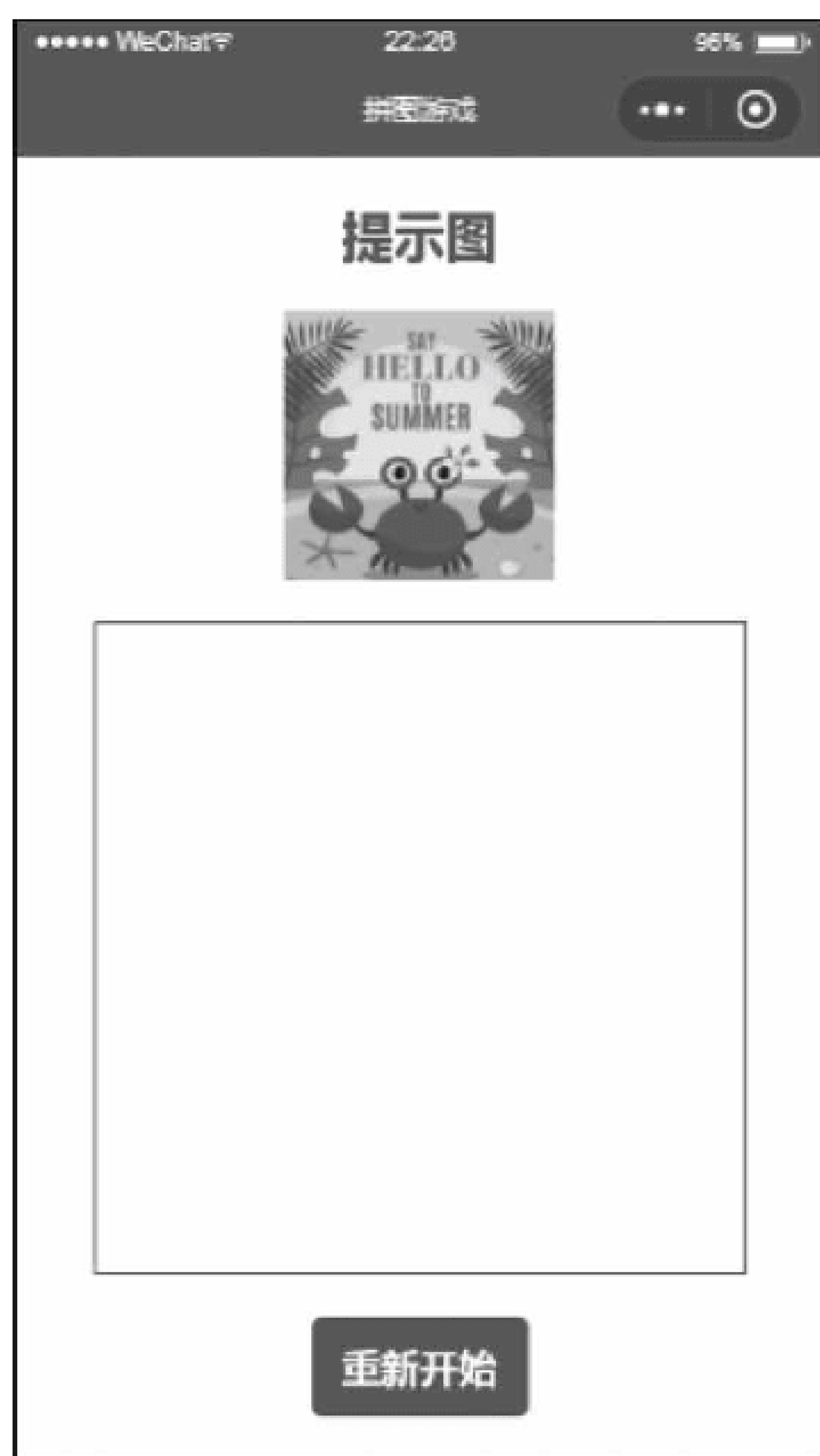
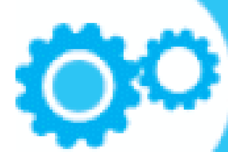


图 12-11 游戏页面效果图

当前效果如图 12-11 所示。

由图可见，此时可以显示完整样式效果。由于尚未获得游戏数据，所以暂时无法根据用户点击的关卡入口显示对应的游戏内容，仅供作为样式参考。

此时页面布局与样式设计就已完成，12.5 节将介绍如何进行逻辑处理。

 12.5 逻辑实现 

12.5.1 首页逻辑



视频讲解

首页主要有两个功能需要实现，一是展示关卡列表，二是点击图片能跳转到游戏页面。

1 关卡列表展示

在 JS 文件的 data 中录入关卡图片的数据信息，这里以 6 个关卡为例。

相关 JS(pages/index/index.js)代码片段如下：

```
1.  Page({
2.    /**
3.     * 页面的初始数据
4.     */
5.    data: {
6.      levels: [
7.        'pic01.jpg',
8.        'pic02.jpg',
9.        'pic03.jpg',
10.       'pic04.jpg',
11.       'pic05.jpg',
12.       'pic06.jpg'
13.     ]
14.   },
15. })
```

接着为关卡对应的< view >组件添加 wx:for 属性循环显示关卡列表数据和图片。修改后的 WXML(pages/index/index.wxml)代码如下：

```
1.  <view class = 'container'>
2.    <!-- 标题 -->
3.    <view class = 'title'>游戏选关</view>
4.
5.    <!-- 关卡列表 -->
6.    <view class = 'levelBox'>
7.      <view class = 'box' wx:for = '{{levels}}' wx:key = 'levels{{index}}'>
8.        <image src = '/images/{{item}}'></image>
9.        <text>第{{index + 1}}关</text>
10.     </view>
11.   </view>
12. </view>
```

此时页面效果如图 12-12 所示。



图 12-12　首页关卡列表展示

2 点击跳转游戏页面



视频讲解

若希望用户点击关卡图片即可实现跳转，需要首先为关卡列表项目添加点击事件。

相关 WXML(pages/index/index.wxml)代码片段修改如下：

```
1.  <view class = 'container'>
2.    <!-- 标题 -->
3.    <view class = 'title'>游戏选关</view>
4.
5.    <!-- 关卡列表 -->
6.    <view class = 'levelBox'>
7.      <view class = 'box' wx:for = '{{levels}}' wx:key =
'levels{{index}}' bindtap = 'chooseLevel' data-level =
'{{item}}'>
8.        <image src = '/images/{{item}}'></image>
9.        <text>第{{index + 1}}关</text>
10.     </view>
11.   </view>
12.
13. </view>
```

上述代码表示为关卡添加了自定义点击事件函数 chooseLevel，并且使用 data-level 属性携带了关卡图片信息。

然后在对应的 index.js 文件中添加 chooseLevel 函数的内容，代码片段如下：

```
1.  Page({
2.    /**
3.     * 自定义函数 -- 游戏选关
4.     */
5.    chooseLevel: function(e) {
6.      let level = e.currentTarget.dataset.level
7.      wx.navigateTo({
8.        url: '../game/game?level=' + level
9.      })
10.   },
11. })
```

此时已经可以点击跳转到 game 页面，并且成功携带了关卡图片数据，但是仍需在 game 页面进行携带数据的接收处理才可显示正确的游戏画面。

12.5.2　游戏页逻辑



视频讲解

游戏页主要有两个功能需要实现，一是显示提示图；二是游戏逻辑实现。

1 显示提示图

在首页逻辑中已经实现了页面跳转并携带了关卡对应的图片信息，现在需要在游戏页面接收关卡信息，并显示对应的图片内容。

相关 JS(pages/game/game.js)代码片段如下：

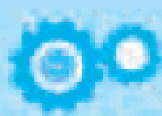
```
1.  Page({
2.    /**
3.     * 生命周期函数 -- 监听页面加载
4.     */
5.    onLoad: function(options) {
6.      //更新图片路径地址
7.      url = '/images/' + options.level
8.      //更新提示图的地址
9.      this.setData({url:url})
10.   },
11. })
```

修改 WXML(pages/game/game.wxml)代码片段如下：

```
1.  <view class='container'>
2.    <!-- 提示图区域 -->
3.    <view class='title'>提示图</view>
4.    <image src='{{url}}'></image>
5.
6.     …
7.  </view>
```

此时重新从首页点击不同的关卡图片跳转就可以发现已经能够正确显示对应的内容了。

运行效果如图 12-13 所示。





(a) 首页关卡列表　　(b) 点击选关效果

图 12-13　首页列表中的选关效果

2 游戏逻辑实现



视频讲解

1）准备工作

首先在 game.js 文件的顶端记录一些游戏初始数据信息。

对应的 JS(pages/game/game.js)代码片段添加如下：

```
1.  //方块的初始位置
2.  var num = [
3.    ['00', '01', '02'],
4.    ['10', '11', '12'],
5.    ['20', '21', '22']
6.  ]
7.  //方块的宽度
8.  var w = 100
9.  //图片的初始地址
10. var url = '/images/pic01.jpg'
11.
12. Page({
13.    …
14. })
```

2）初始化拼图画面



视频讲解

传统做法是随机抽取画面中的任意两个方块，然后交换彼此的位置，在进行足够多次数的交换后基本可以实现随机打乱的效果。但是这种方法有一个弊端，就是有时候会陷入无解的死局，例如图 12-14 所示的效果。

00	01	02
10	11	12
20	22	21

图 12-14　无解的游戏效果

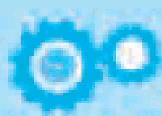
因此可以考虑从空白方块的所在位置入手，每次随机让它和

周围的邻近方块交换位置，这样可以通过方块反向移动回到最初始状态（确保本局有解），并且在交换足够多的次数后也可以实现随机打乱的效果。

在 game.js 文件中添加 shuffle 函数，用于重新开始游戏。

对应的 JS(pages/game/game.js)代码片段添加如下：

```
1.  Page({
2.    /**
3.     * 自定义函数 -- 随机打乱方块顺序
4.     */
5.    shuffle: function() {
6.      //先令所有方块回归初始位置
7.      num = [
8.        ['00', '01', '02'],
9.        ['10', '11', '12'],
10.       ['20', '21', '22']
11.     ]
12.     //记录当前空白方块的行和列
13.     var row = 2
14.     var col = 2
15.     //打乱方块顺序 100 次
16.     for (var i = 0; i < 100; i++) {
17.       //随机产生其中一个方向：上(0)、下(1)、左(2)、右(3)
18.       var direction = Math.round(Math.random() * 3)
19.       //上：0
20.       if (direction == 0) {
21.         //空白方块不在最上面一行
22.         if (row != 0) {
23.           //交换位置
24.           num[row][col] = num[row - 1][col]
25.           num[row - 1][col] = '22'
26.           //更新空白方块的行
27.           row -= 1
28.         }
29.       }
30.       //下：1
31.       else if (direction == 1) {
32.         //空白方块不在最下面一行
33.         if (row != 2) {
34.           //交换位置
35.           num[row][col] = num[row + 1][col]
36.           num[row + 1][col] = '22'
37.           //更新空白方块的行
38.           row += 1
39.         }
40.       }
41.       //左：2
42.       else if (direction == 2) {
43.         //空白方块不在最左侧
44.         if (col != 0) {
45.           //交换位置
46.           num[row][col] = num[row][col - 1]
47.           num[row][col - 1] = '22'
48.           //更新空白方块的列
49.           col -= 1
```

```
50.         }
51.       }
52.       //右: 3
53.       else if (direction == 3) {
54.         //空白方块不在最右侧
55.         if (col != 2) {
56.           //交换位置
57.           num[row][col] = num[row][col + 1]
58.           num[row][col + 1] = '22'
59.           //更新空白方块的列
60.           col += 1
61.         }
62.       }
63.     }
64.   },
65. })
```

上述代码表示使用 for 循环进行了 100 次打乱，开发者可以根据自己的需求更改循环次数。每次使用 Math.random()方法从上、下、左、右 4 个方向中随机产生一个方向，之后如果符合条件则交换空白方块和图片方块的位置。

然后在 game.js 中添加自定义函数 drawCanvas，用于将打乱后的图片方块绘制到画布上。

对应的 JS(pages/game/game.js)代码片段添加如下：

```
1.  Page({
2.    /**
3.     * 自定义函数--绘制画布内容
4.     */
5.    drawCanvas: function() {
6.      let ctx = this.ctx
7.  
8.      //清空画布
9.      ctx.clearRect(0, 0, 300, 300)
10. 
11.     //使用双重 for 循环绘制 3×3 的拼图
12.     for (var i = 0; i < 3; i++) {
13.       for (var j = 0; j < 3; j++) {
14.         if (num[i][j] != '22') {
15.           //获取行和列
16.           var row = parseInt(num[i][j] / 10)
17.           var col = num[i][j] % 10
18.           //绘制方块
19.           ctx.drawImage(url, col * w, row * w, w, w, j * w, i * w, w, w)
20.         }
21.       }
22.     }
23.     ctx.draw()
24.   },
25. })
```

最后在 game.js 的 onLoad 函数中调用自定义函数 shuffle 和 drawCanvas。

对应的 JS(pages/game/game.js)代码片段添加如下：

```
1.  Page({
```

```
2.      * 生命周期函数 -- 监听页面加载
3.      * /
4.    onLoad: function(options) {
5.      //更新图片路径地址
6.      url = '/images/' + options.level
7.      //更新提示图的地址
8.      this.setData({url:url})
9.      //创建画布上下文
10.     this.ctx = wx.createCanvasContext('myCanvas')
11.     //打乱方块顺序
12.     this.shuffle()
13.     //绘制画布内容
14.     this.drawCanvas()
15.   },
16. })
```



图 12-15　随机打乱效果

当前效果如图 12-15 所示。

3）移动被点击的方块



视频讲解

修改 game. wxml 页面中的画布组件(<canvas>)，为其绑定触摸事件。

WXML(pages/game/game. wxml)代码修改后如下：

```
1.  < view class = 'container'>
2.    <!-- 提示图区域(代码略) -->
3.
4.    <!-- 游戏画布 -->
5.    < canvas canvas - id = 'myCanvas' bindtouchstart = 'touchBox'></canvas >
6.
7.    <!--“重新开始”按钮(代码略) -->
8.  </view>
```

在 game. js 文件中添加自定义函数 touchBox，用于实现图片方块的移动。

对应的 JS(pages/game/game. js)代码片段添加如下：

```
1.  Page({
2.    / **
3.      *  自定义函数 -- 监听点击方块事件
4.      * /
5.    touchBox: function(e) {
6.      // 如果游戏已经成功,不做任何操作
7.      if (this.data.isWin) {
8.        // 终止本函数
9.        return
10.      }
11.
12.      // 获取被点击方块的坐标 x 和 y
13.      var x  =  e.changedTouches[0].x
14.      var y  =  e.changedTouches[0].y
15.      // console.log('x:' + x + ',y:' + y)
16.
17.      // 换算成行和列
```

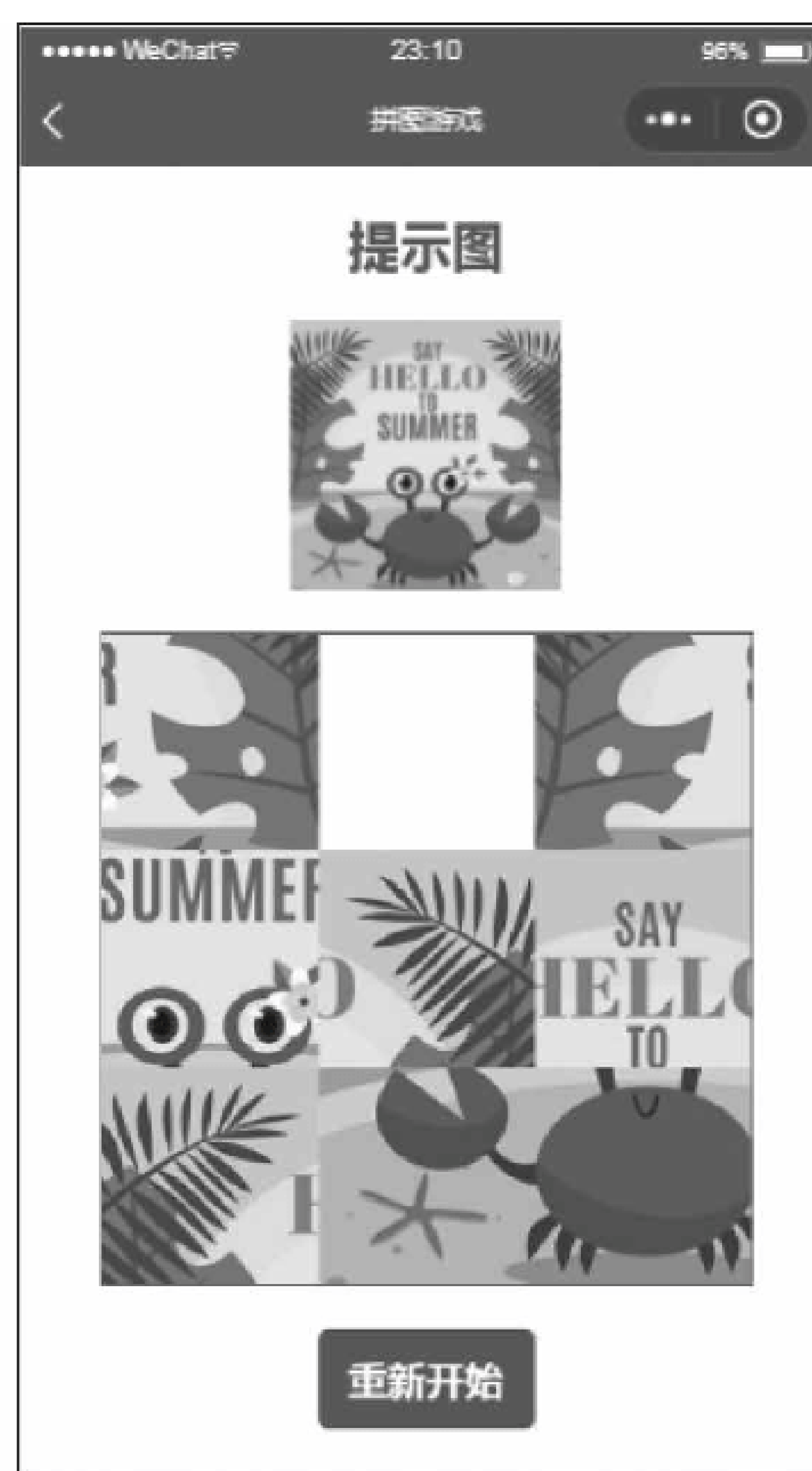
```
18.         var row = parseInt(y / w)
19.         var col = parseInt(x / w)
20.
21.         // 如果点击的不是空白位置
22.         if (num[row][col] != '22') {
23.           // 尝试移动方块
24.           this.moveBox(row, col)
25.
26.           // 重新绘制画布内容
27.           this.drawCanvas()
28.
29.           // 判断游戏是否成功
30.           if (this.isWin()) {
31.             // 在画面上绘制提示语句
32.             let ctx = this.ctx
33.
34.             // 绘制完整图片
35.             ctx.drawImage(url, 0, 0)
36.
37.             // 绘制文字
38.             ctx.setFillStyle('#e64340')
39.             ctx.setTextAlign('center')
40.             ctx.setFontSize(60)
41.             ctx.fillText('游戏成功', 150, 150)
42.             ctx.draw()
43.           }
44.         }
45.       },
46.       /**
47.        * 自定义函数--移动被点击的方块
48.        */
49.       moveBox: function(i, j) {
50.         //情况 1:如果被点击的方块不在最上方,检查可否上移
51.         if (i > 0) {
52.           //如果方块上方是空白
53.           if (num[i - 1][j] == '22') {
54.             //交换方块与空白的位置
55.             num[i - 1][j] = num[i][j]
56.             num[i][j] = '22'
57.             return
58.           }
59.         }
60.
61.         //情况 2:如果被点击的方块不在最下方,检查可否下移
62.         if (i < 2) {
63.           //如果方块下方是空白
64.           if (num[i + 1][j] == '22') {
65.             //交换方块与空白的位置
66.             num[i + 1][j] = num[i][j]
67.             num[i][j] = '22'
68.             return
69.           }
```

```
70.         }
71.
72.         //情况 3:如果被点击的方块不在最左边,检查可否左移
73.         if (j > 0) {
74.           //如果方块左边是空白
75.           if (num[i][j - 1] == '22') {
76.             //交换方块与空白的位置
77.             num[i][j - 1] = num[i][j]
78.             num[i][j] = '22'
79.             return
80.           }
81.         }
82.
83.         //情况 4:如果被点击的方块不在最右边,检查可否右移
84.         if (j < 2) {
85.           //如果方块右边是空白
86.           if (num[i][j + 1] == '22') {
87.             //交换方块与空白的位置
88.             num[i][j + 1] = num[i][j]
89.             num[i][j] = '22'
90.             return
91.           }
92.         }
93.     },
94.   })
```

当前效果如图 12-16 所示。



(a) 移动前



(b) 移动后

图 12-16　移动被点击的方块

3 判断游戏成功



视频讲解

首先在 game.js 文件的 data 中添加初始数据 isWin，用于标记游戏成功与否。

对应的 JS(pages/game/game.js)代码片段添加如下：

```
1.  Page({
2.    /**
3.     * 页面的初始数据
4.     */
5.    data: {
6.      isWin:false
7.    },
8.  })
```

在上述代码中 isWin 为 false 表示游戏尚未成功，当成功时会重置为 true。

在 game.js 文件中添加自定义函数 isWin，用于判断游戏是否已经成功。

对应的 JS(pages/game/game.js)代码片段添加如下：

```
1.  Page({
2.    /**
3.     * 自定义函数 -- 判断游戏是否成功
4.     */
5.    isWin: function() {
6.      //使用双重 for 循环遍历整个数组
7.      for (var i = 0; i < 3; i++) {
8.        for (var j = 0; j < 3; j++) {
9.          //如果有方块位置不对
10.         if (num[i][j] != i * 10 + j) {
11.           //返回 false,表示游戏尚未成功
12.           return false
13.         }
14.       }
15.     }
16.     //更新游戏成功状态
17.     this.setData({isWin:true})
18.     //返回 true,表示游戏成功
19.     return true
20.   },
21. })
```

然后修改 game.js 中的 touchBox 函数，要求被触发时追加对游戏成功状态的判断。

对应的 JS(pages/game/game.js)代码片段修改如下：

```
1.  Page({
2.    /**
3.     * 自定义函数 -- 点击方块
4.     */
5.    touchBox: function(e) {
6.      //如果游戏已经成功,不做任何操作
7.      if(this.data.isWin){
8.        return
9.      }
```

```
10.         //获取被点击的坐标 x 和 y(代码略)
11.         //换算成行和列(代码略)
12.
13.         //如果点击的不是空白位置
14.         if (num[row][col] != '22') {
15.           //尝试移动方块(代码略)
16.           //重新绘制画布内容(代码略)
17.
18.           //判断游戏是否成功
19.           if (this.isWin()) {
20.             let ctx = this.ctx
21.             //绘制完整图片
22.             ctx.drawImage(url, 0, 0)
23.             //绘制文字提示
24.             ctx.setFillStyle('#E64340')
25.             ctx.setTextAlign('center')
26.             ctx.setFontSize(60)
27.             ctx.fillText('游戏成功', 150, 150)
28.             ctx.draw()
29.           }
30.         }
31.     },
32. })
```



图 12-17 游戏成功效果

游戏成功效果如图 12-17 所示。

4 重新开始游戏



视频讲解

修改 game.wxml 代码，为“重新开始”按钮追加自定义函数的点击事件。

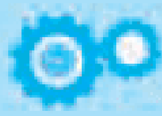
WXML(pages/game/game.wxml)代码片段修改如下：

```
1.  <view class='container'>
2.    ...
3.
4.    <!-- "重新开始"按钮 -->
5.    <button type='warn' bindtap='restartGame'>重新开始</button>
6.  </view>
```

在 game.js 文件中添加 restartGame 函数，用于重新开始游戏。

对应的 JS(pages/game/game.js)代码片段添加如下：

```
1.  Page({
2.    /**
3.     * 自定义函数--重新开始游戏
4.     */
5.    restartGame: function() {
6.      //更新游戏成功状态
7.      this.setData({isWin:false})
8.      //打乱方块顺序
9.      this.shuffle()
10.     //绘制画布内容
11.     this.drawCanvas()
```

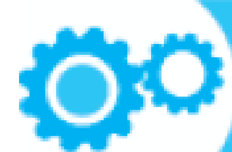


```
12.     },
13.  })
```

此时页面效果如图 12-18 所示。



图 12-18 重新开始游戏效果



12.6 完整代码展示



12.6.1 应用文件代码展示

1 app.json 代码展示

app.json 文件的完整代码如下：

```
1.  {
2.    "pages": [
3.      "pages/index/index",
4.      "pages/game/game"
5.    ],
6.    "window": {
7.      "navigationBarBackgroundColor": "#E64340",
8.      "navigationBarTitleText": "拼图游戏"
9.    }
10. }
```

2 app.wxss 代码展示

app.wxss 文件的完整代码如下：

```
1.  /* 页面容器样式 */
2.  .container {
3.    height: 100vh;
```

```
4.     color: #E64340;
5.     font-weight: bold;
6.     display: flex;
7.     flex-direction: column;
8.     align-items: center;
9.     justify-content: space-evenly;
10. }
11. /* 顶端标题样式 */
12. .title {
13.     font-size: 18pt;
14. }
```

12.6.2 页面文件代码展示

1 首页代码展示

WXML 文件(pages/index/index.wxml)的完整代码如下：

```
1. <view class='container'>
2.     <!-- 标题 -->
3.     <view class='title'>游戏选关</view>
4.
5.     <!-- 关卡列表 -->
6.     <view class='levelBox'>
7.         <view class='box' wx:for='{{levels}}' wx:key='levels{{index}}' bindtap='chooseLevel'
data-level='{{item}}'>
8.             <image src='/images/{{item}}'></image>
9.             <text>第{{index+1}}关</text>
10.         </view>
11.     </view>
12.
13. </view>
```

WXSS 文件(pages/index/index.wxss)的完整代码如下：

```
1. /* 关卡列表区域 */
2. .levelBox{
3.     width: 100%;
4. }
5. /* 单个关卡区域 */
6. .box{
7.     width: 50%;
8.     float: left;
9.     margin: 25rpx 0;
10.     display: flex;
11.     flex-direction: column;
12.     align-items: center;
13. }
14. /* 选关图片 */
15. image{
16.     width: 260rpx;
17.     height: 260rpx;
18. }
```

JS 文件(pages/index/index.js)的完整代码如下：

```
1.  Page({
2.    /**
3.     * 页面的初始数据
4.     */
5.    data: {
6.      levels: [
7.        'pic01.jpg',
8.        'pic02.jpg',
9.        'pic03.jpg',
10.       'pic04.jpg',
11.       'pic05.jpg',
12.       'pic06.jpg'
13.     ]
14.   },
15.
16.   /**
17.    * 自定义函数--游戏选关
18.    */
19.   chooseLevel: function(e) {
20.     let level = e.currentTarget.dataset.level
21.     wx.navigateTo({
22.       url: '../game/game?level=' + level
23.     })
24.   },
25. })
```

2 游戏页代码展示

WXML 文件(pages/game/game.wxml)的完整代码如下：

```
1.  <view class='container'>
2.    <!-- 提示图区域 -->
3.    <view class='title'>提示图</view>
4.    <image src='{{url}}'></image>
5.
6.    <!-- 游戏画布 -->
7.    <canvas canvas-id='myCanvas' bindtouchstart='touchBox'></canvas>
8.
9.    <!-- "重新开始"按钮 -->
10.   <button type='warn' bindtap='restartGame'>重新开始</button>
11. </view>
```

WXSS 文件(pages/game/game.wxss)的完整代码如下：

```
1.  /* 提示图样式 */
2.  image {
3.    width: 250rpx;
4.    height: 250rpx;
5.  }
6.
7.  /* 画布样式 */
8.  canvas {
9.    border: 1rpx solid;
10.   width: 300px;
11.   height: 300px;
12. }
```

JS 文件(pages/game/game.js)的完整代码如下：

```
1.  //方块的初始位置
2.  var num = [
3.    ['00', '01', '02'],
4.    ['10', '11', '12'],
5.    ['20', '21', '22']
6.  ]
7.  //方块的宽度
8.  var w = 100
9.  //图片的初始地址
10. var url = '/images/pic01.jpg'
11.
12. Page({
13.   /**
14.    * 页面的初始数据
15.    */
16.   data: {
17.     isWin:false
18.   },
19.   /**
20.    * 自定义函数--绘制画布内容
21.    */
22.   drawCanvas: function() {
23.     let ctx = this.ctx
24.
25.     //清空画布
26.     ctx.clearRect(0, 0, 300, 300)
27.
28.     //使用双重 for 循环绘制 3x3 的拼图
29.     for (var i = 0; i < 3; i++) {
30.       for (var j = 0; j < 3; j++) {
31.         if (num[i][j] != '22') {
32.           //获取行和列
33.           var row = parseInt(num[i][j] / 10)
34.           var col = num[i][j] % 10
35.           //绘制方块
36.           ctx.drawImage(url, col * w, row * w, w, w, j * w, i * w, w, w)
37.         }
38.       }
39.     }
40.     ctx.draw()
41.   },
42.   /**
43.    * 自定义函数--随机打乱方块顺序
44.    */
45.   shuffle: function() {
46.     //先令所有方块回归初始位置
47.     num = [
48.       ['00', '01', '02'],
49.       ['10', '11', '12'],
50.       ['20', '21', '22']
51.     ]
52.     //记录当前空白方块的行和列
53.     var row = 2
54.     var col = 2
```

```
55.     //打乱方块顺序 100 次
56.     for (var i = 0; i < 100; i++) {
57.       //随机产生其中一个方向：上(0)、下(1)、左(2)、右(3)
58.       var direction = Math.round(Math.random() * 3)
59.       //上：0
60.       if (direction == 0) {
61.         //空白方块不在最上面一行
62.         if (row != 0) {
63.           //交换位置
64.           num[row][col] = num[row - 1][col]
65.           num[row - 1][col] = '22'
66.           //更新空白方块的行
67.           row -= 1
68.         }
69.       }
70.       //下：1
71.       else if (direction == 1) {
72.         //空白方块不在最下面一行
73.         if (row != 2) {
74.           //交换位置
75.           num[row][col] = num[row + 1][col]
76.           num[row + 1][col] = '22'
77.           //更新空白方块的行
78.           row += 1
79.         }
80.       }
81.       //左：2
82.       else if (direction == 2) {
83.         //空白方块不在最左侧
84.         if (col != 0) {
85.           //交换位置
86.           num[row][col] = num[row][col - 1]
87.           num[row][col - 1] = '22'
88.           //更新空白方块的列
89.           col -= 1
90.         }
91.       }
92.       //右：3
93.       else if (direction == 3) {
94.         //空白方块不在最右侧
95.         if (col != 2) {
96.           //交换位置
97.           num[row][col] = num[row][col + 1]
98.           num[row][col + 1] = '22'
99.           //更新空白方块的列
100.          col += 1
101.        }
102.      }
103.    }
104.  },
105.
106.  /**
107.   * 自定义函数--点击方块
108.   */
109.  touchBox: function(e) {
```

```
110.    //如果游戏已经成功,不做任何操作
111.    if(this.data.isWin){
112.      return
113.    }
114.    //获取被点击的坐标 x 和 y
115.    var x = e.changedTouches[0].x
116.    var y = e.changedTouches[0].y
117.    //换算成行和列
118.    var row = parseInt(y / w)
119.    var col = parseInt(x / w)
120.    //如果点击的不是空白位置
121.    if (num[row][col] != '22') {
122.      //尝试移动方块
123.      this.moveBox(row, col)
124.      //重新绘制画布内容
125.      this.drawCanvas()
126.      //判断游戏是否成功
127.      if (this.isWin()) {
128.        let ctx = this.ctx
129.        //绘制完整图片
130.        ctx.drawImage(url, 0, 0)
131.        //绘制文字提示
132.        ctx.setFillStyle('#E64340')
133.        ctx.setTextAlign('center')
134.        ctx.setFontSize(60)
135.        ctx.fillText('游戏成功', 150, 150)
136.        ctx.draw()
137.      }
138.    }
139.  },
140.
141.  /**
142.   * 自定义函数--移动被点击的方块
143.   */
144.  moveBox: function(i, j) {
145.    //情况 1: 如果被点击的方块不在最上方,检查可否上移
146.    if (i > 0) {
147.      //如果方块上方是空白
148.      if (num[i - 1][j] == '22') {
149.        //交换方块与空白的位置
150.        num[i - 1][j] = num[i][j]
151.        num[i][j] = '22'
152.        return
153.      }
154.    }
155.
156.    //情况 2: 如果被点击的方块不在最下方,检查可否下移
157.    if (i < 2) {
158.      //如果方块下方是空白
159.      if (num[i + 1][j] == '22') {
160.        //交换方块与空白的位置
161.        num[i + 1][j] = num[i][j]
162.        num[i][j] = '22'
163.        return
164.      }
```

```
165.     }
166.
167.     //情况 3: 如果被点击的方块不在最左边,检查可否左移
168.     if (j > 0) {
169.       //如果方块左边是空白
170.       if (num[i][j - 1] == '22') {
171.         //交换方块与空白的位置
172.         num[i][j - 1] = num[i][j]
173.         num[i][j] = '22'
174.         return
175.       }
176.     }
177.
178.     //情况 4: 如果被点击的方块不在最右边,检查可否右移
179.     if (j < 2) {
180.       //如果方块右边是空白
181.       if (num[i][j + 1] == '22') {
182.         //交换方块与空白的位置
183.         num[i][j + 1] = num[i][j]
184.         num[i][j] = '22'
185.         return
186.       }
187.     }
188.   },
189.
190.   /**
191.    * 自定义函数 -- 判断游戏是否成功
192.    */
193.   isWin: function() {
194.     //使用双重 for 循环遍历整个数组
195.     for (var i = 0; i < 3; i++) {
196.       for (var j = 0; j < 3; j++) {
197.         //如果有方块位置不对
198.         if (num[i][j] != i * 10 + j) {
199.           //返回 false,表示游戏尚未成功
200.           return false
201.         }
202.       }
203.     }
204.     //更新游戏成功状态
205.     this.setData({isWin:true})
206.     //返回 true,表示游戏成功
207.     return true
208.   },
209.
210.   /**
211.    * 自定义函数 -- 重新开始游戏
212.    */
213.   restartGame: function() {
214.     //更新游戏成功状态
215.     this.setData({isWin:false})
216.     //打乱方块顺序
217.     this.shuffle()
218.     //绘制画布内容
219.     this.drawCanvas()
```

```
220.    },
221.
222.    /**
223.     * 生命周期函数--监听页面加载
224.     */
225.    onLoad: function(options) {
226.      //更新图片路径地址
227.      url = '/images/' + options.level
228.      //更新提示图的地址
229.      this.setData({url:url})
230.      //创建画布上下文
231.      this.ctx = wx.createCanvasContext('myCanvas')
232.      //打乱方块顺序
233.      this.shuffle()
234.      //绘制画布内容
235.      this.drawCanvas()
236.    },
237. })
```



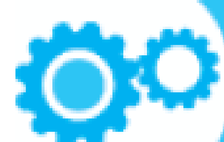
第13章 Chapter 13

小程序游戏·推箱子游戏

在学习了<canvas>(画布)组件和小程序界面 API 中绘图的相关用法以后,读者不妨尝试制作简易的推箱子小游戏。

本章学习目标

- 综合应用所学知识创建完整的推箱子游戏项目;
- 熟练掌握<canvas>组件和绘图 API。



13.1 需求分析





视频讲解

本项目一共需要两个页面,即首页和游戏页面,其中首页用于呈现关卡菜单,点击对应难度的关卡后进入游戏画面。

13.1.1 首页功能需求

首页功能需求如下:

(1) 首页需要包含标题和关卡列表。

(2) 关卡至少要有 4 个关卡选项,每个关卡显示预览图片和第几关。

(3) 点击关卡列表可以打开对应的游戏画面。

13.1.2 游戏页功能需求

游戏页功能需求如下:

(1) 游戏页面需要显示第几关、游戏画面、方向键和"重新开始"按钮。

(2) 点击方向键可以使游戏主角自行移动或推动箱子前进。

(3) 游戏画面由 8×8 的小方块组成,主要包括地板、围墙、箱子、游戏主角和目的地。

(4) 点击"重新开始"按钮可以将箱子和游戏主角回归初始位置并重新开始游戏。



13.2 项目创建





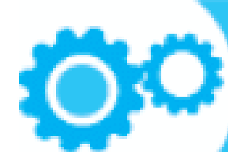
视频讲解

本项目创建选择空白文件夹 boxGame,效果如图 13-1 所示。

单击"新建"按钮完成项目创建,然后准备手动修改页面配置文件。



图 13-1 小程序项目填写效果示意图

 13.3 页面配置 

13.3.1 创建页面文件

项目创建完毕后，在根目录中会生成文件夹 pages 用于存放页面文件。一般来说首页默认命名为 index，表示小程序运行的第一个页面；其他页面名称可以自定义。本项目有两个页面文件，需要创建 index（首页页面）和 game（游戏页面）。

具体操作如下：

（1）将 app.json 文件内 pages 属性中的“pages/logs/logs”改成“pages/game/game”。

（2）按快捷键 Ctrl+S 保存修改后会在 pages 文件夹下自动生成 game 目录。

13.3.2 删除和修改文件

具体操作如下：

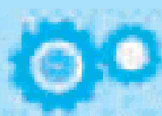
（1）删除 utils 文件夹及其内部所有内容。

（2）删除 pages 文件夹下的 logs 目录及其内部所有内容。

（3）删除 index.wxml 和 index.wxss 中的全部代码。

（4）删除 index.js 中的全部代码，并且输入关键词 page 找到第二个选项按回车键让其自动补全函数（如图 13-2 所示）。

（5）删除 app.wxss 中的全部代码。



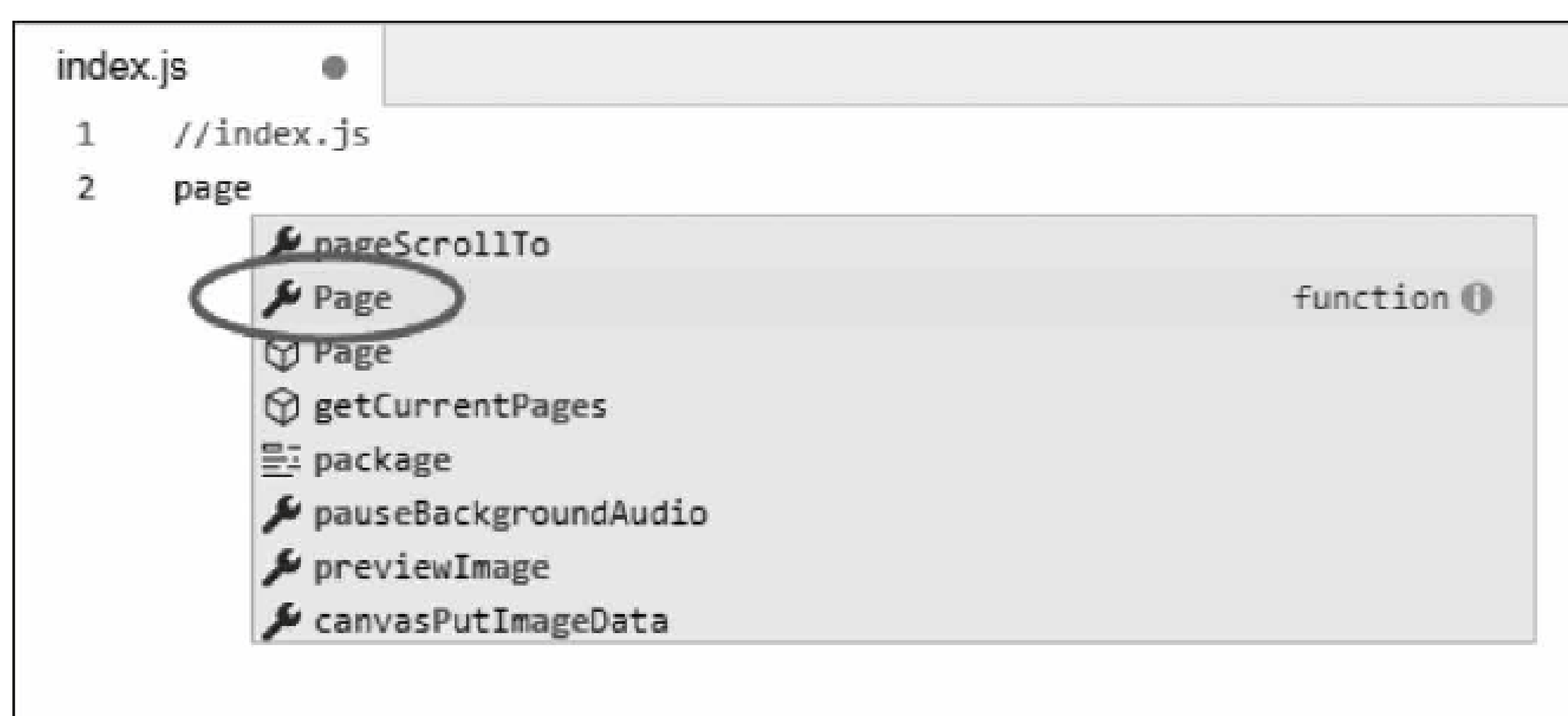


图 13-2 输入关键词创建 Page 函数

(6) 删除 app.js 中的全部代码,并且输入关键词 app 找到第一个选项按回车键让其自动补全函数(如图 13-3 所示)。

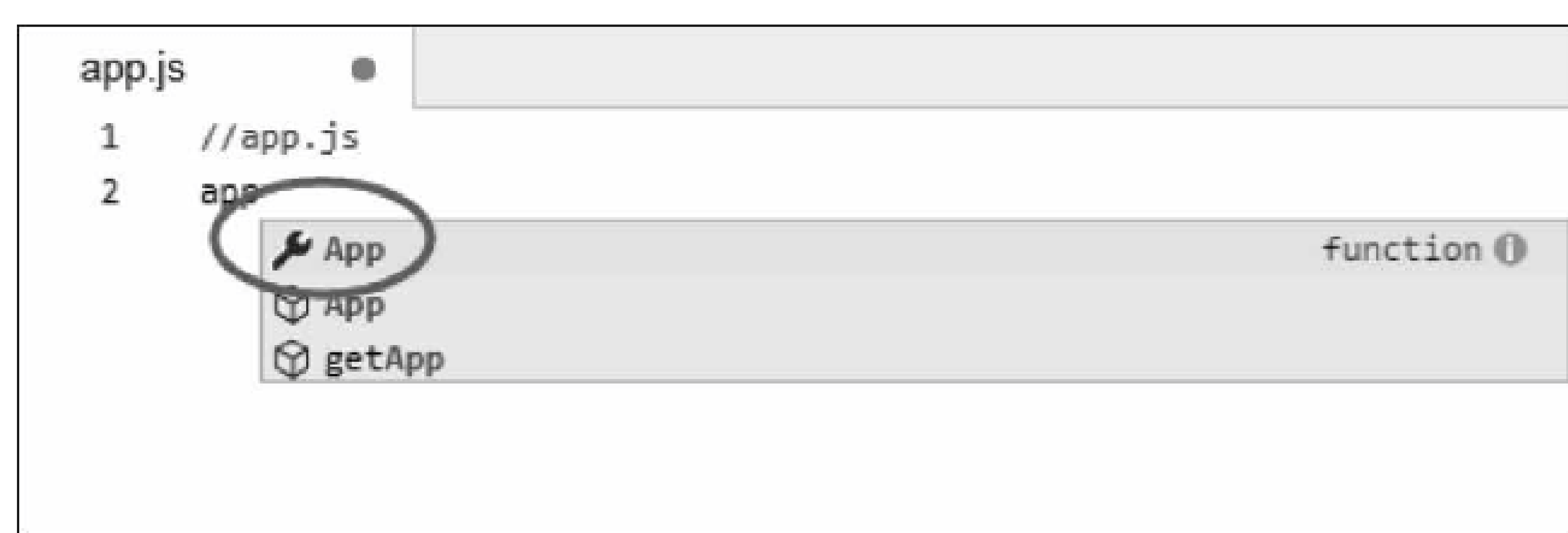


图 13-3 输入关键词创建 App 函数

13.3.3 创建其他文件

接下来创建其他自定义文件,本项目还需要以下两个文件夹。

- images: 用于存放图片素材;
- utils: 用于存放公共 JS 文件。

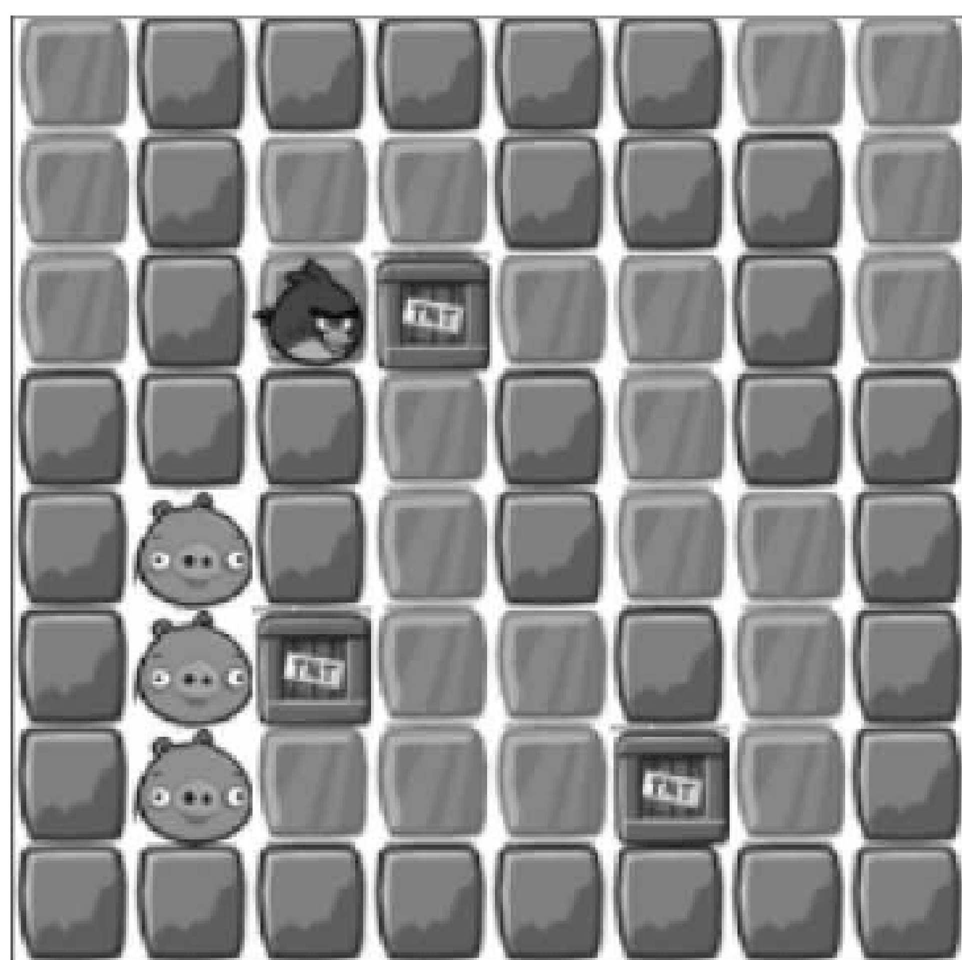
单击目录结构左上角的+号创建文件夹,并分别命名为 images 和 utils。

1 添加图片文件

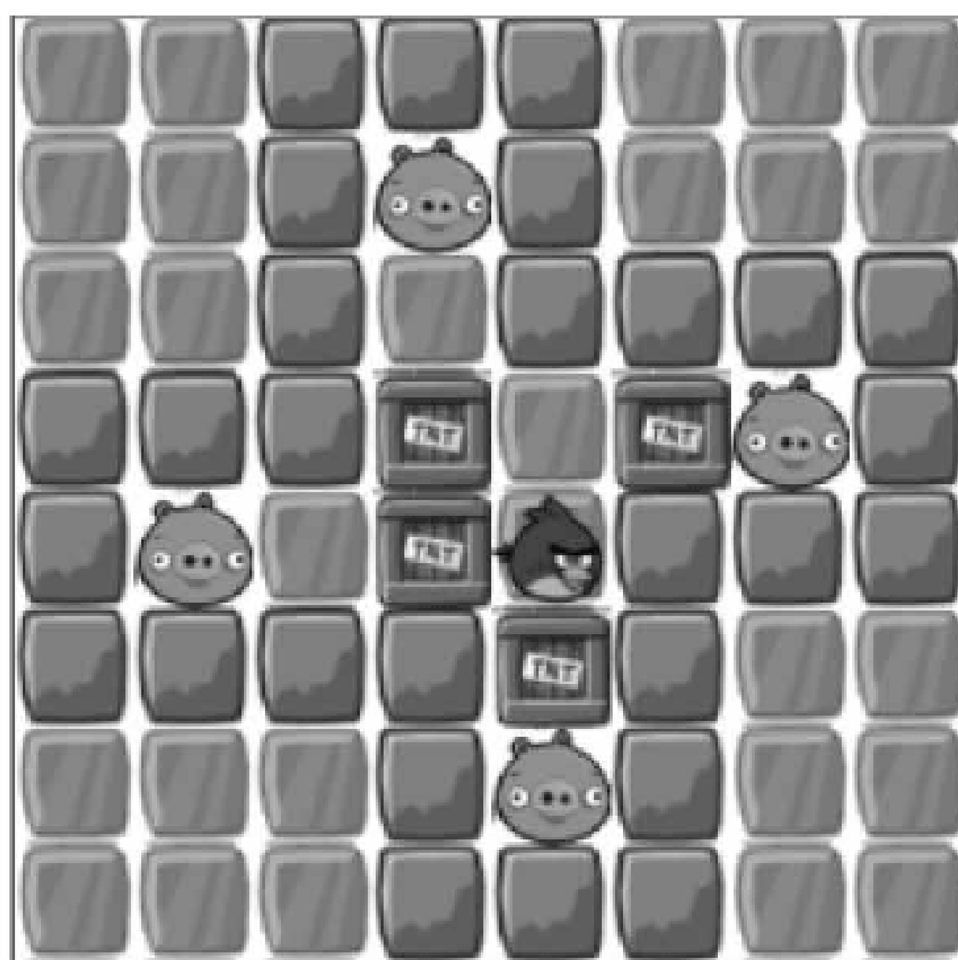
本项目将在首页中用到 4 幅关卡图片,图片素材如图 13-4 所示。

右击目录结构中的 images 文件夹,选择"硬盘打开",将图片复制、粘贴进去。

此外还需要在游戏页面中用到 5 个游戏图标素材,如图 13-5 所示。

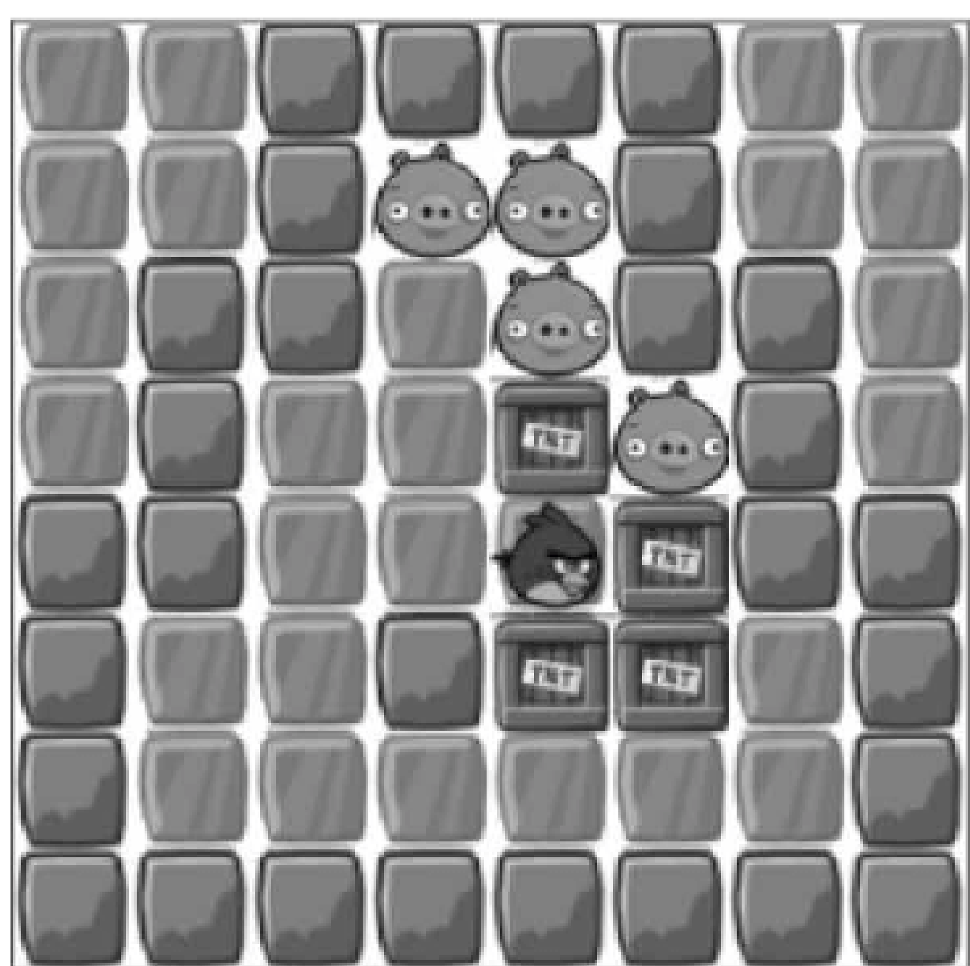


(a) level01.png

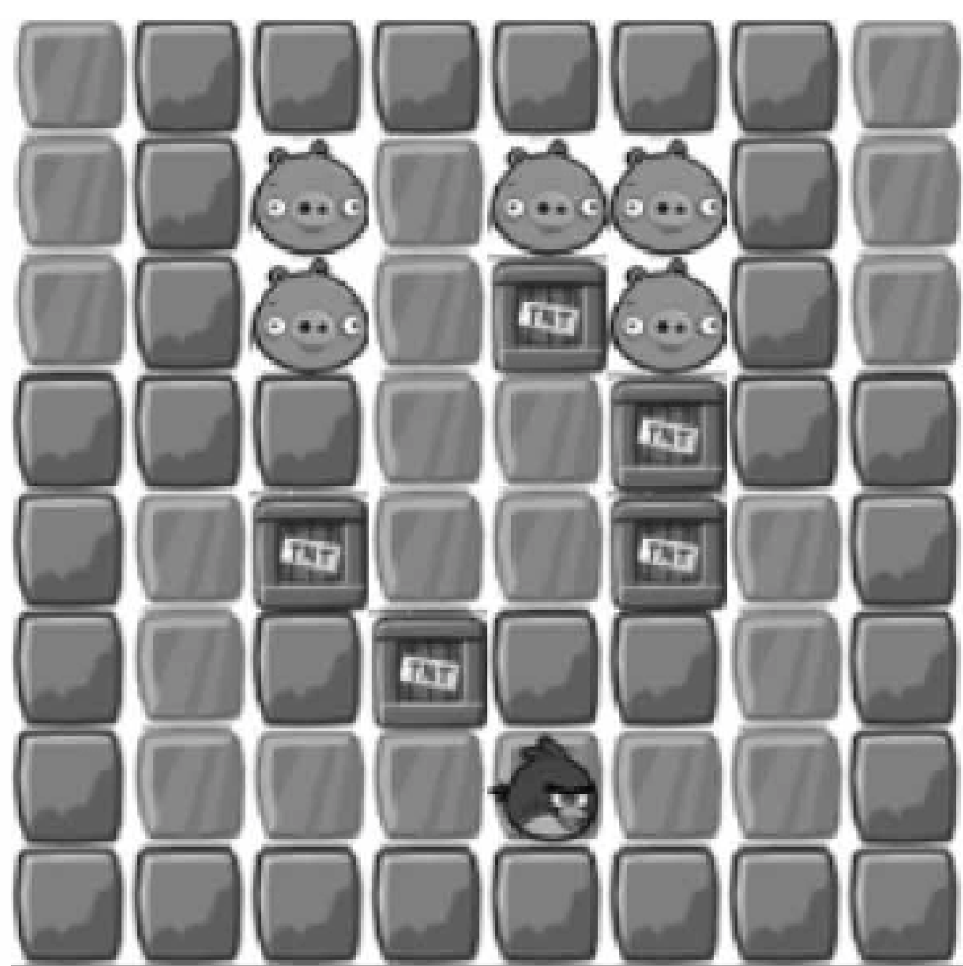


(b) level02.png

图 13-4 关卡图片素材展示



(c) level03.png

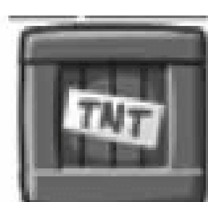


(d) level04.png

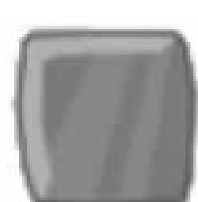
图 13-4 （续）



(a) bird.png



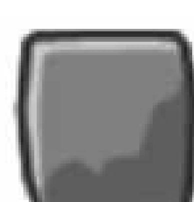
(b) box.png



(c) ice.png



(d) pig.png



(e) stone.png

图 13-5 游戏方块图标素材展示

右击目录结构中的 images 文件夹，选择“硬盘打开”，新建二级文件夹 icons，然后将全部图标素材复制、粘贴进去。

2 创建公共 JS 文件

右击 utils 文件夹，选择“新建”→JS，输入 data 后按回车键创建公共文件 data. js，如图 13-6 所示。

全部完成后的目录结构如图 13-7 所示。



图 13-6 新建 JS 文件

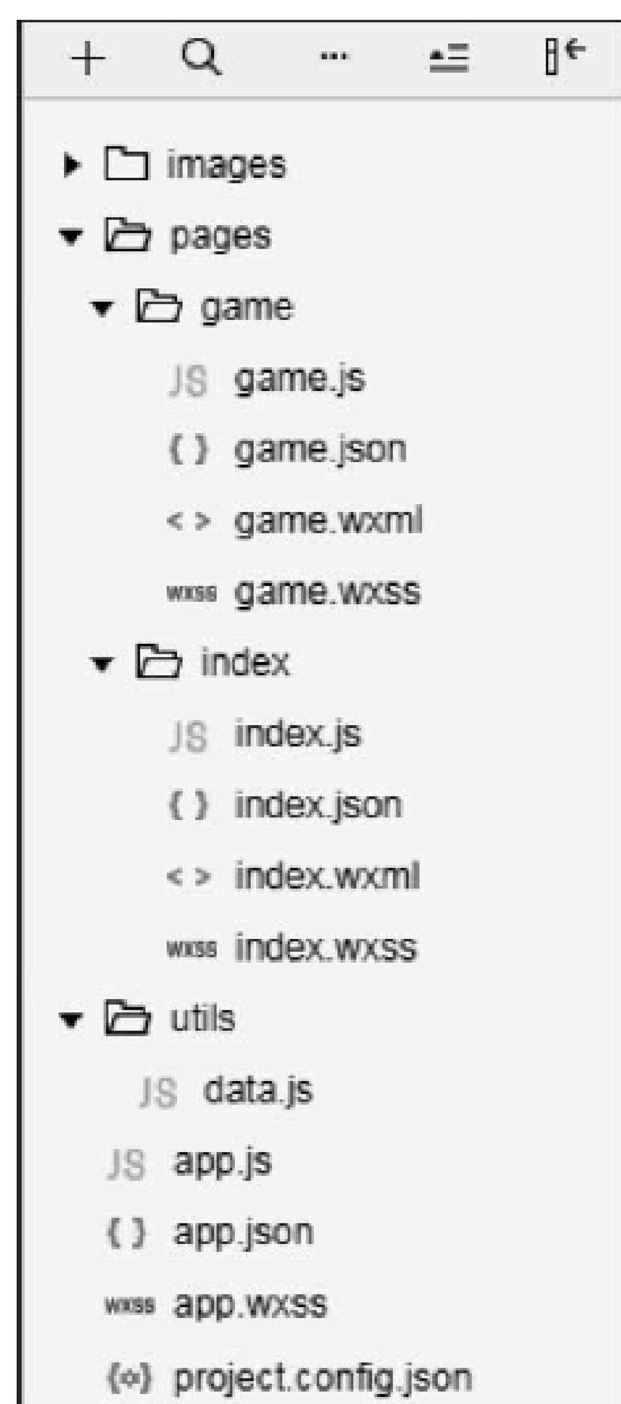
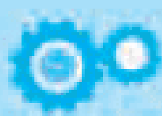


图 13-7 全部文件创建完成

此时文件配置就全部完成，13. 4 节将正式进行页面布局和样式设计。



13.4 视图设计



13.4.1 导航栏设计



视频讲解

小程序默认导航栏是黑底白字的效果，可以通过在 app. json 中对 window 属性进行重新配置来自定义导航栏效果。更改后的 app. json 文件代码如下：

```
1.  {
2.    "pages": [代码略],
3.    "window": {
4.      "navigationBarBackgroundColor": "#E64340",
5.      "navigationBarTitleText": "推箱子游戏"
6.    }
7.  }
```

上述代码可以更改导航栏背景色为珊瑚红色、字体为白色，效果如图 13-8 所示。

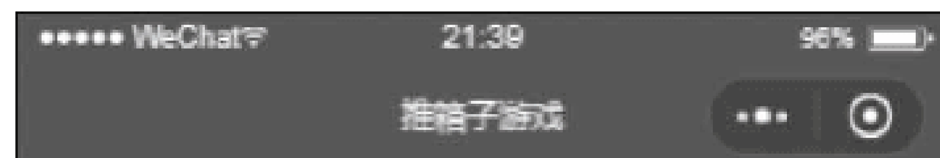


图 13-8 自定义导航栏效果

13.4.2 页面设计



视频讲解

1 公共样式设计

首先在 app. wxss 中设置页面容器和顶端标题的公共样式，代码如下：

```
1.  /* 页面容器样式 */
2.  .container {
3.    height: 100vh;
4.    color: #E64340;
5.    font-weight: bold;
6.    display: flex;
7.    flex-direction: column;
8.    align-items: center;
9.    justify-content: space-evenly;
10. }
11. /* 顶端标题样式 */
12. .title {
13.   font-size: 18pt;
14. }
```



视频讲解

2 首页设计

首页主要包含两部分内容，即标题和关卡列表，页面设计如图 13-9 所示。

计划使用如下组件。

- 顶端标题：<view>容器；
- 关卡列表：<view>容器，内部使用数组循环。

WXML(pages/index/index. wxml)代码如下：

```
1.  <view class='container'>
2.    <!-- 标题 -->
3.    <view class='title'>游戏选关</view>
4.
```

```
5.     <!-- 关卡列表 -->
6.       < view class = 'levelBox'>
7.       < view class = 'box'>
8.         < image src = '/images/level01.png'></image >
9.         < text >第 1 关</text >
10.      </view >
11.    </view >
12.
13.  </view >
```

相关 WXSS(pages/index/index.wxss)代码片段如下：

```
1.  /* 关卡列表区域 */
2.  .levelBox {
3.    width: 100%;
4.  }
5.
6.  /* 单个关卡区域 */
7.  .box {
8.    width: 50%;
9.    float: left;
10.   margin: 20rpx 0;
11.   display: flex;
12.   flex-direction: column;
13.   align-items: center;
14. }
15.
16. /* 选关图片 */
17. image {
18.   width: 300rpx;
19.   height: 300rpx;
20. }
```

当前效果如图 13-10 所示。

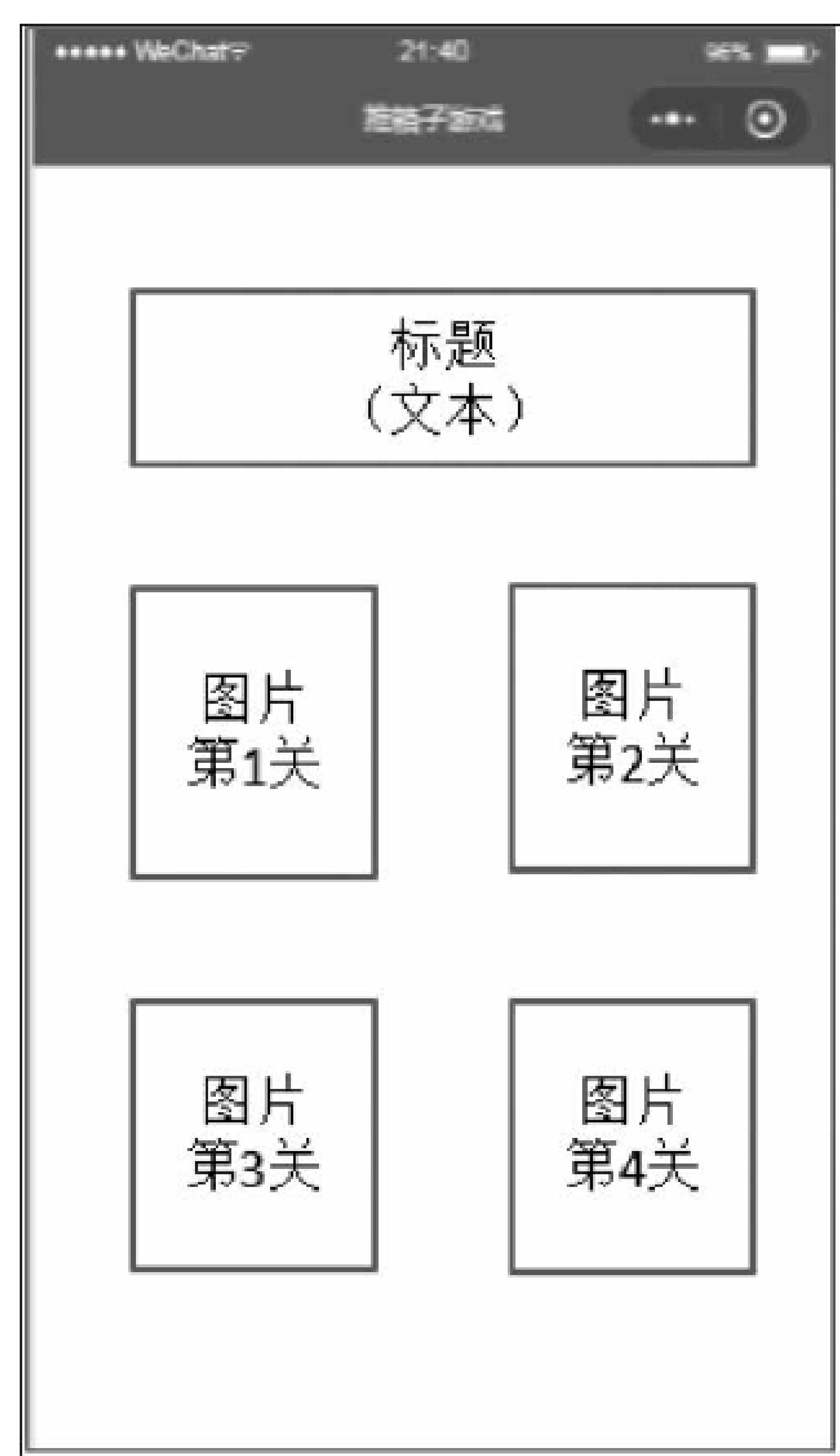


图 13-9 首页设计图

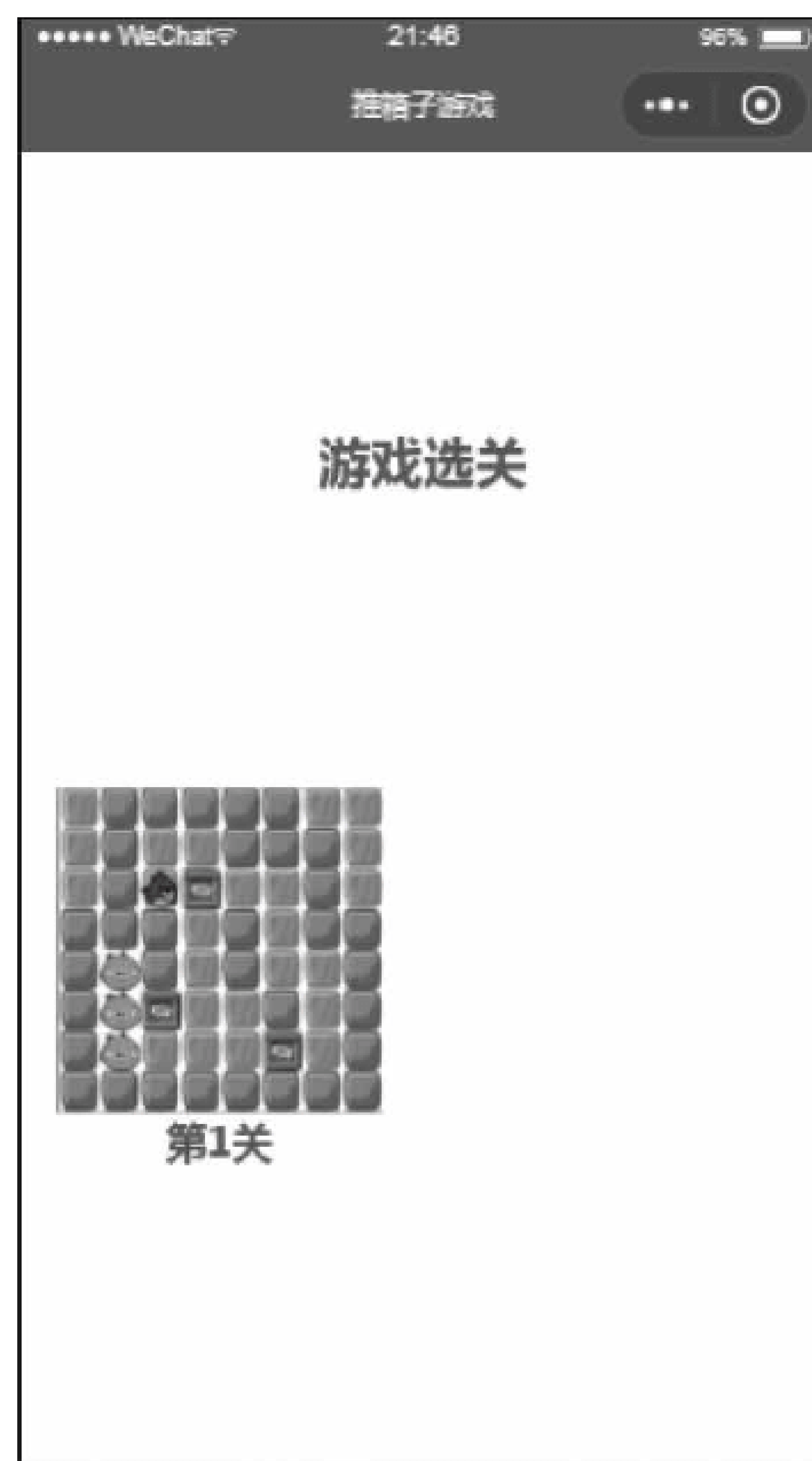


图 13-10 首页效果图



由图可见，此时可以显示标题和一个临时关卡。由于尚未获得关卡数据，所以暂时无法显示完整的关卡列表，仅供作为样式参考。

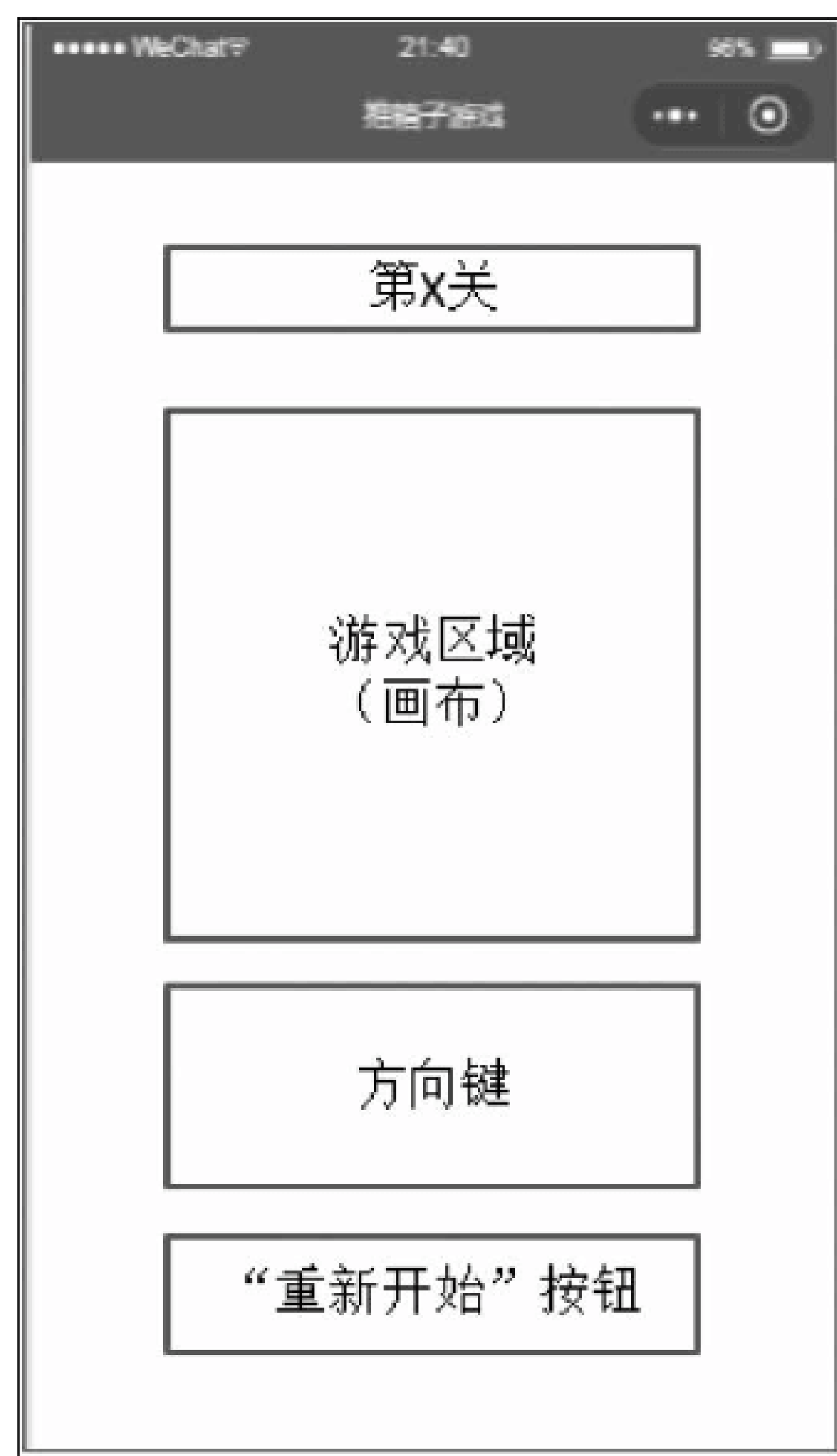


图 13-11　游戏页面设计图

3 游戏页面设计


视频讲解

游戏页面需要用户点击首页的关卡列表，然后在新窗口中打开该页面。游戏页面包括游戏关卡标题、游戏画面、方向键和“重新开始”按钮，页面设计如图 13-11 所示。

由于暂时没有做点击跳转的逻辑设计，所以可以在开发工具顶端选择“普通编译”下的“添加编译模式”，并携带临时测试参数 level=0，如图 13-12 所示。

此时预览就可以直接显示 game 页面了，设计完毕后再改回“普通编译”模式即可重新显示首页。

计划使用如下组件。

- <view>：整体容器和顶端标题；
- <canvas>：游戏画布；

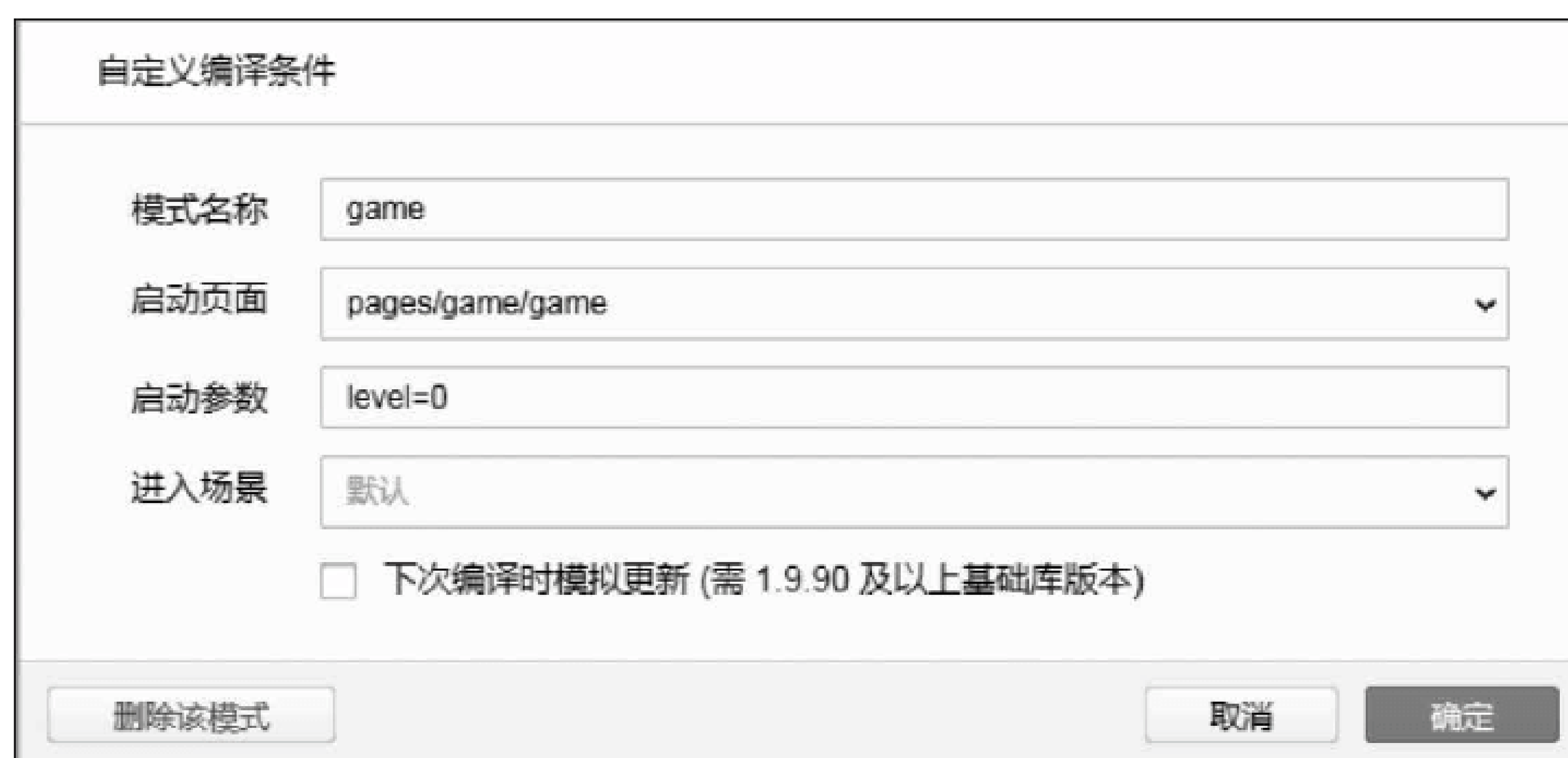


图 13-12　添加 game 页面的编译模式

- <button>：4 个方向键和 1 个“重新开始”按钮。

WXML(pages/game/game.wxml)代码如下：

```
1.  <view class = 'container'>
2.    <!-- 关卡提示 -->
3.    <view class = 'title'>第 1 关</view>
4.
5.    <!-- 游戏画布 -->
6.    <canvas canvas - id = 'myCanvas'></canvas>
7.
8.    <!-- 方向键 -->
9.    <view class = 'btnBox'>
10.     <button type = 'warn'>↑</button>
11.     <view>
12.       <button type = 'warn'>←</button>
13.       <button type = 'warn'>↓</button>
14.       <button type = 'warn'>→</button>
15.     </view>
16.   </view>
```

```
17.
18.   <!--"重新开始"按钮 -->
19.   <button type='warn'>重新开始</button>
20. </view>
```

WXSS(pages/game/game.wxss)代码如下：

```
1.  /* 游戏画布样式 */
2.  canvas {
3.    border: 1rpx solid;
4.    width: 320px;
5.    height: 320px;
6.  }
7.
8.  /* 方向键按钮整体区域 */
9.  .btnBox {
10.   display: flex;
11.   flex-direction: column;
12.   align-items: center;
13. }
14.
15. /* 方向键按钮第二行 */
16. .btnBox view {
17.   display: flex;
18.   flex-direction: row;
19. }
20.
21. /* 所有方向键按钮 */
22. .btnBox button {
23.   width: 90rpx;
24.   height: 90rpx;
25. }
26.
27. /* 所有按钮样式 */
28. button {
29.   margin: 10rpx;
30. }
```

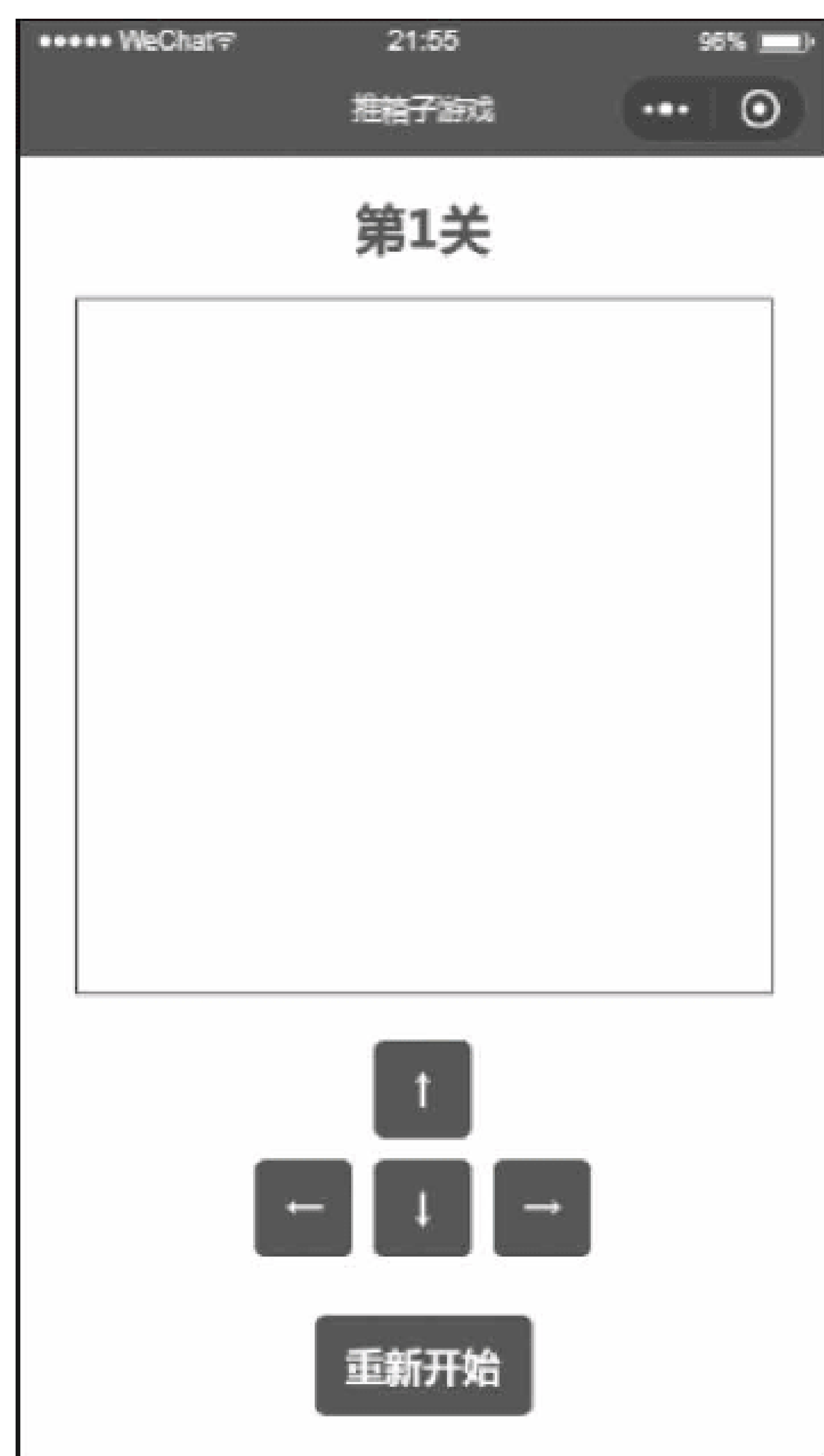
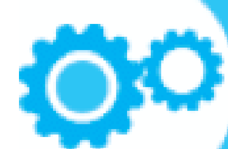


图 13-13　游戏页面效果图

当前效果如图 13-13 所示。

由图可见，此时可以显示完整样式效果。由于尚未获得游戏数据，所以暂时无法根据用户点击的关卡入口显示对应的游戏内容，仅供作为样式参考。

此时页面布局与样式设计就已完成，13.5 节将介绍如何进行逻辑处理。

 13.5　逻辑实现

13.5.1　公共逻辑



视频讲解

在公共 JS 文件(utils/data.js)中配置游戏地图的数据，代码如下：

```
1.  //=============================================================
2.  //地图数据 map1～map4
3.  //地图数据：1 为墙、2 为路、3 为终点、4 为箱子、5 为人物、0 为墙的外围
4.  //=============================================================
```

```
5.  //关卡 1
6.  var map1 = [
7.    [0, 1, 1, 1, 1, 1, 0, 0],
8.    [0, 1, 2, 2, 1, 1, 1, 0],
9.    [0, 1, 5, 4, 2, 2, 1, 0],
10.   [1, 1, 1, 2, 1, 2, 1, 1],
11.   [1, 3, 1, 2, 1, 2, 2, 1],
12.   [1, 3, 4, 2, 2, 1, 2, 1],
13.   [1, 3, 2, 2, 2, 4, 2, 1],
14.   [1, 1, 1, 1, 1, 1, 1, 1]
15. ]
16. //关卡 2
17. var map2 = [
18.   [0, 0, 1, 1, 1, 0, 0, 0],
19.   [0, 0, 1, 3, 1, 0, 0, 0],
20.   [0, 0, 1, 2, 1, 1, 1, 1],
21.   [1, 1, 1, 4, 2, 4, 3, 1],
22.   [1, 3, 2, 4, 5, 1, 1, 1],
23.   [1, 1, 1, 1, 4, 1, 0, 0],
24.   [0, 0, 0, 1, 3, 1, 0, 0],
25.   [0, 0, 0, 1, 1, 1, 0, 0]
26.
27. ]
28. //关卡 3
29. var map3 = [
30.   [0, 0, 1, 1, 1, 1, 0, 0],
31.   [0, 0, 1, 3, 3, 1, 0, 0],
32.   [0, 1, 1, 2, 3, 1, 1, 0],
33.   [0, 1, 2, 2, 4, 3, 1, 0],
34.   [1, 1, 2, 2, 5, 4, 1, 1],
35.   [1, 2, 2, 1, 4, 4, 2, 1],
36.   [1, 2, 2, 2, 2, 2, 2, 1],
37.   [1, 1, 1, 1, 1, 1, 1, 1]
38. ]
39. //关卡 4
40. var map4 = [
41.   [0, 1, 1, 1, 1, 1, 1, 0],
42.   [0, 1, 3, 2, 3, 3, 1, 0],
43.   [0, 1, 3, 2, 4, 3, 1, 0],
44.   [1, 1, 1, 2, 2, 4, 1, 1],
45.   [1, 2, 4, 2, 2, 4, 2, 1],
46.   [1, 2, 1, 4, 1, 1, 2, 1],
47.   [1, 2, 2, 2, 5, 2, 2, 1],
48.   [1, 1, 1, 1, 1, 1, 1, 1]
49. ]
```

这里分别使用 map1～map4 代表 4 个不同关卡的地图数据，以二维数组的形式存放。当前地图均由 8×8 的方格组成，每个位置的数字代表对应的图标素材。

当前地图数据和图片素材仅供参考，开发者也可以自行修改游戏布局和图片。

然后需要在 data.js 中使用 module.exports 语句暴露数据出口，代码片段如下：

```
1.  module.exports = {
2.    maps: [map1, map2, map3, map4]
3.  }
```

现在就完成了公共逻辑处理的部分。

最后需要在 game 页面的 JS 文件顶端引用公共 JS 文件，引用代码如下：

```
var data = require('../../utils/data.js')                    //引用公共 JS 文件
```

需要注意小程序在这里暂时还不支持绝对路径引用，只能使用相对路径。

13.5.2 首页逻辑

首页主要有两个功能需要实现，一是展示关卡列表；二是点击图片能跳转到游戏页面。

1 关卡列表展示



视频讲解

在 JS 文件的 data 中录入关卡图片的数据信息，这里以 4 个关卡为例。

相关 JS(pages/index/index.js)代码片段如下：

```
1.  Page({
2.    /**
3.     * 页面的初始数据
4.     */
5.    data: {
6.      levels: [
7.        'level01.png',
8.        'level02.png',
9.        'level03.png',
10.       'level04.png'
11.     ]
12.   },
13. })
```

接着为关卡对应的< view >组件添加 wx:for 属性循环显示关卡列表数据和图片。

修改后的 WXML(pages/index/index.wxml)代码如下：

```
1.  <view class = 'container'>
2.    <!-- 标题 -->
3.    <view class = 'title'>游戏选关</view>
4.
5.    <!-- 关卡列表 -->
6.    <view class = 'levelBox'>
7.      <view class = 'box' wx:for = '{{levels}}' wx:key =
'levels{{index}}'>
8.        <image src = '/images/{{item}}'></image>
9.        <text>第{{index + 1}}关</text>
10.     </view>
11.   </view>
12.
13. </view>
```

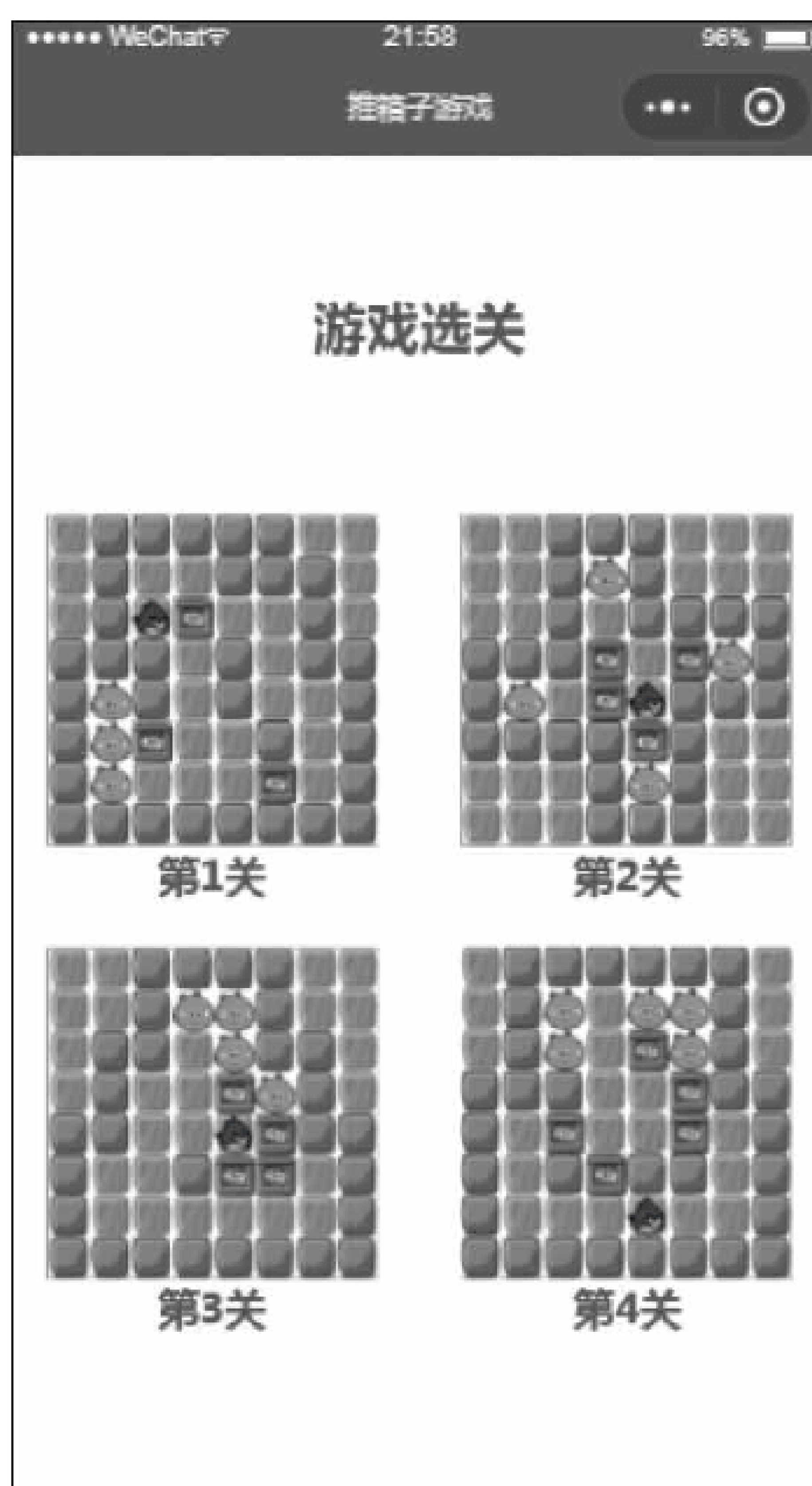


图 13-14 首页关卡列表展示

此时页面效果如图 13-14 所示。

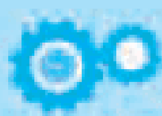
2 点击跳转游戏页面



视频讲解

若希望用户点击关卡图片即可实现跳转，需要首先为关卡列表项目添加点击事件。

相关 WXML(pages/index/index.wxml)代码片段修改如下：



```
1.  <view class='container'>
2.    <!-- 标题 -->
3.    <view class='title'>游戏选关</view>
4.
5.    <!-- 关卡列表 -->
6.    <view class='levelBox'>
7.      <view class='box' wx:for='{{levels}}' wx:key='levels{{index}}' bindtap='chooseLevel' 
data-level='{{index}}'>
8.        <image src='/images/{{item}}'></image>
9.        <text>第{{index+1}}关</text>
10.     </view>
11.   </view>
12.
13. </view>
```

上述代码表示为关卡添加了自定义点击事件函数 chooseLevel，并且使用 data-level 属性携带了关卡图片下标信息。

然后在对应的 index.js 文件中添加 chooseLevel 函数的内容，代码片段如下：

```
1.  Page({
2.    /**
3.     * 自定义函数--游戏选关
4.     */
5.    chooseLevel: function(e) {
6.      let level = e.currentTarget.dataset.level
7.      wx.navigateTo({
8.        url: '../game/game?level=' + level
9.      })
10.   },
11. })
```

此时已经可以点击跳转到 game 页面，并且成功携带了关卡图片数据，但是仍需在 game 页面进行携带数据的接收处理才可显示正确的游戏画面。

13.5.3 游戏页逻辑

游戏页主要有以下几个功能需要实现：

- 显示当前是第几关；
- 游戏地图的绘制；
- 4 个方向键可以移动游戏主角；
- 点击“重新开始”按钮可以使游戏地图还原成最初状态。

1 显示当前第几关


视频讲解

在首页逻辑中已经实现了页面跳转并携带了关卡对应的图片信息，现在需要在游戏页面接收关卡信息，并显示对应的图片内容。

相关 JS(pages/game/game.js)代码片段如下：

```
1.  Page({
2.    /**
3.     * 页面的初始数据
4.     */
5.    data: {
```

```
6.        level: 1
7.      },
8.      /**
9.        * 生命周期函数 -- 监听页面加载
10.       */
11.     onLoad: function(options) {
12.       //获取关卡
13.       let level = options.level
14.       //更新页面关卡标题
15.       this.setData({
16.         level: parseInt(level) + 1
17.       })
18.     },
19.   })
```

修改 WXML(pages/game/game.wxml)代码片段如下：

```
1.  <view class = 'container'>
2.    <!-- 关卡提示 -->
3.    <view class = 'title'>第{{level}}关</view>
4.
5.     …
6.  </view>
```

此时重新从首页点击不同的关卡图片跳转就可以发现已经能够正确显示对应的内容了。

运行效果如图13-15所示。

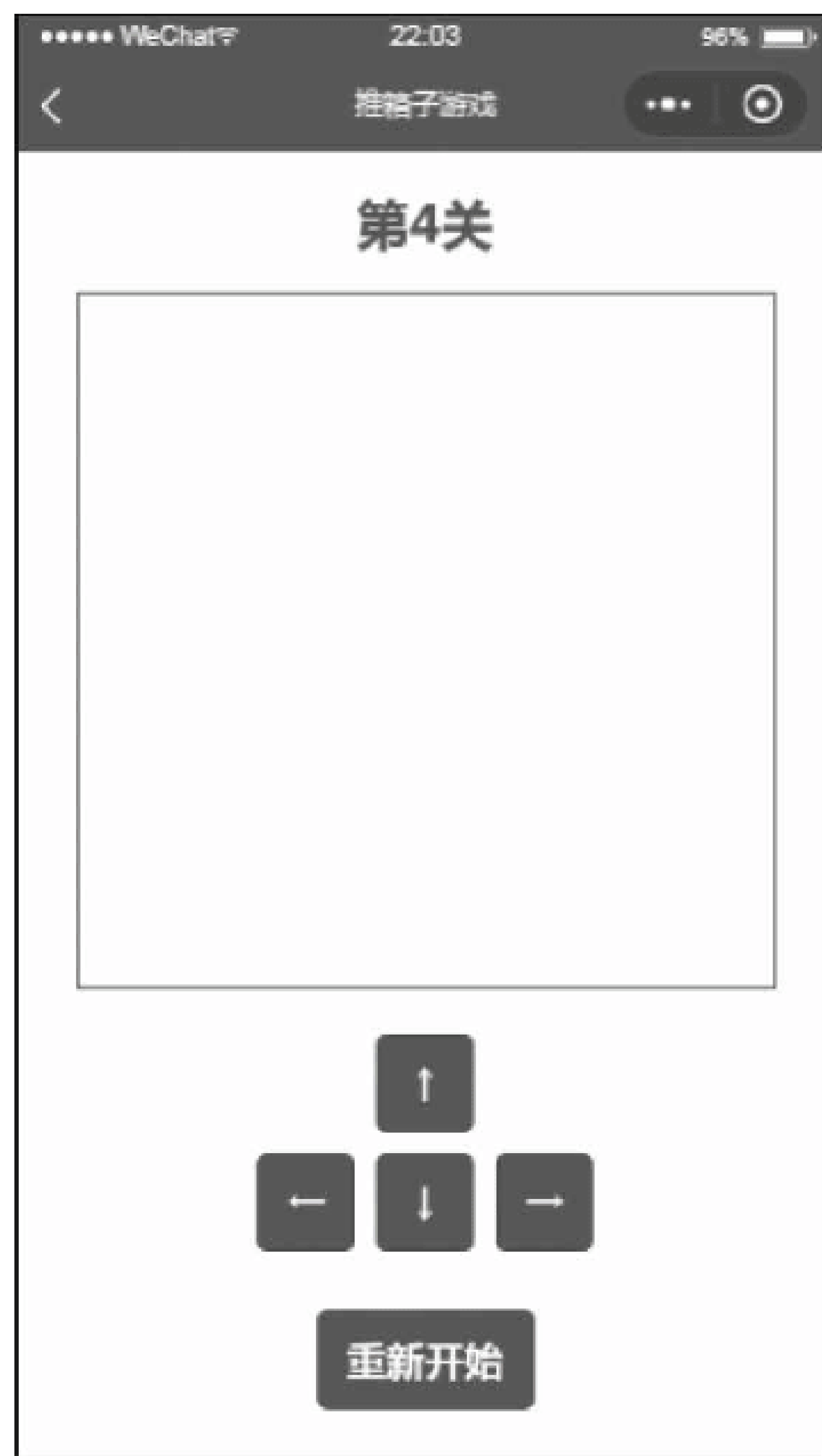


图 13-15 首页列表中的选关效果

2 游戏逻辑实现



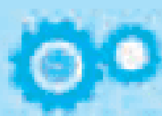
视频讲解

1）准备工作

首先在game.js文件的顶端记录一些游戏初始数据信息。

对应的JS(pages/game/game.js)代码片段添加如下：

```
1.  //地图图层数据
2.  var map = [
3.    [0, 0, 0, 0, 0, 0, 0, 0],
4.    [0, 0, 0, 0, 0, 0, 0, 0],
5.    [0, 0, 0, 0, 0, 0, 0, 0],
6.    [0, 0, 0, 0, 0, 0, 0, 0],
7.    [0, 0, 0, 0, 0, 0, 0, 0],
8.    [0, 0, 0, 0, 0, 0, 0, 0],
9.    [0, 0, 0, 0, 0, 0, 0, 0],
10.   [0, 0, 0, 0, 0, 0, 0, 0]
11. ]
12. //箱子图层数据
13. var box = [
14.   [0, 0, 0, 0, 0, 0, 0, 0],
15.   [0, 0, 0, 0, 0, 0, 0, 0],
16.   [0, 0, 0, 0, 0, 0, 0, 0],
17.   [0, 0, 0, 0, 0, 0, 0, 0],
18.   [0, 0, 0, 0, 0, 0, 0, 0],
19.   [0, 0, 0, 0, 0, 0, 0, 0],
20.   [0, 0, 0, 0, 0, 0, 0, 0],
21.   [0, 0, 0, 0, 0, 0, 0, 0]
```



```
22. ]
23. //方块的宽度
24. var w = 40
25. //初始化游戏主角(小鸟)的行与列
26. var row = 0
27. var col = 0
28.
29. Page({
30.     …
31. })
```

2）初始化游戏画面


视频讲解

首先需要根据当前是第几关读取对应的游戏地图信息，并更新到游戏初始数据中。

在 game.js 文件中添加 initMap 函数，用于初始化游戏地图数据。

对应的 JS(pages/game/game.js)代码片段添加如下：

```
1.  Page({
2.    /**
3.     * 自定义函数 -- 初始化地图数据
4.     */
5.    initMap: function(level) {
6.      //读取原始的游戏地图数据
7.      let mapData = data.maps[level]
8.      //使用双重 for 循环记录地图数据
9.      for (var i = 0; i < 8; i++) {
10.       for (var j = 0; j < 8; j++) {
11.         box[i][j] = 0
12.         map[i][j] = mapData[i][j]
13.
14.         if (mapData[i][j] == 4) {
15.           box[i][j] = 4
16.           map[i][j] = 2
17.         } else if (mapData[i][j] == 5) {
18.           map[i][j] = 2
19.           //记录小鸟的当前行和列
20.           row = i
21.           col = j
22.         }
23.       }
24.     }
25.   },
26. })
```

上述代码首先从公共函数文件 data.js 中读取对应关卡的游戏地图数据，然后使用双重 for 循环对每一块地图数据进行解析，并更新当前游戏的初始地图数据、箱子数据以及游戏主角(小鸟)的所在位置。

然后在 game.js 中添加自定义函数 drawCanvas，用于将地图信息绘制到画布上。

对应的 JS(pages/game/game.js)代码片段添加如下：

```
1.  Page({
2.    /**
3.     * 自定义函数 -- 绘制地图
4.     */
5.    drawCanvas: function() {
```

```
6.        let ctx = this.ctx
7.        //清空画布
8.        ctx.clearRect(0, 0, 320, 320)
9.        //使用双重 for 循环绘制 8x8 的地图
10.       for (var i = 0; i < 8; i++) {
11.         for (var j = 0; j < 8; j++) {
12.           //默认是道路
13.           let img = 'ice'
14.           if (map[i][j] == 1) {
15.             img = 'stone'
16.           } else if (map[i][j] == 3) {
17.             img = 'pig'
18.           }
19.
20.           //绘制地图
21.           ctx.drawImage('/images/icons/' + img + '.png', j * w, i * w, w, w)
22.
23.           if (box[i][j] == 4) {
24.             //叠加绘制箱子
25.             ctx.drawImage('/images/icons/box.png', j * w, i * w, w, w)
26.           }
27.         }
28.       }
29.
30.       //叠加绘制小鸟
31.       ctx.drawImage('/images/icons/bird.png', col * w, row * w, w, w)
32.
33.       ctx.draw()
34.     },
35.   })
```

最后在 game.js 的 onLoad 函数中创建画布上下文,并依次调用自定义函数 initMap 和 drawCanvas。对应的 JS(pages/game/game.js)代码片段添加如下:

```
1.  Page({
2.      * 生命周期函数 -- 监听页面加载
3.      */
4.    onLoad: function(options) {
5.      //获取关卡
6.      … 代码略 …
7.      //更新页面关卡标题
8.      … 代码略 …
9.      //创建画布上下文
10.     this.ctx = wx.createCanvasContext('myCanvas')
11.     //初始化地图数据
12.     this.initMap(level)
13.     //绘制画布内容
14.     this.drawCanvas()
15.   },
16. })
```

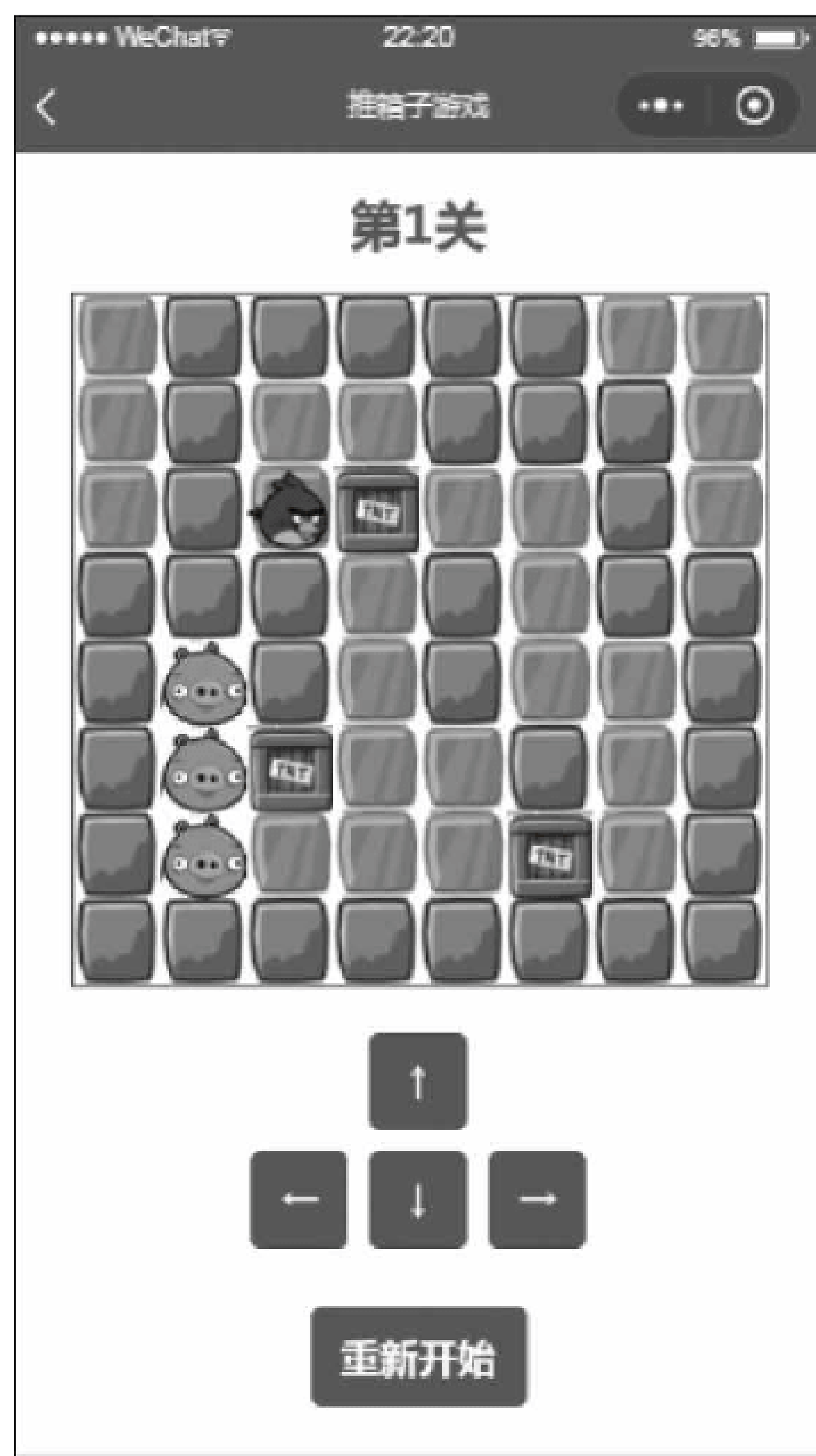


图 13-16 绘制地图效果

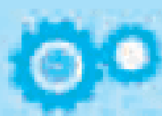
当前效果如图 13-16 所示。

3) 方向键逻辑实现

修改 game.wxml 页面中的 4 个方向键 <button>,为其绑定点击事件。



视频讲解



WXML(pages/game/game.wxml)代码修改后如下：

```
1.  <view class = 'container'>
2.    <!-- 关卡提示(代码略) -->
3.
4.    <!-- 游戏画布(代码略) -->
5.
6.    <!-- 方向键 -->
7.    <view class = 'btnBox'>
8.      <button type = 'warn' bindtap = 'up'>↑</button>
9.      <view>
10.       <button type = 'warn' bindtap = 'left'>←</button>
11.       <button type = 'warn' bindtap = 'down'>↓</button>
12.       <button type = 'warn' bindtap = 'right'>→</button>
13.     </view>
14.   </view>
15.
16.   <!--"重新开始"按钮(代码略) -->
17. </view>
```

在 game.js 文件中添加自定义函数 up、down、left 和 right，分别用于实现游戏主角(小鸟)在上、下、左、右 4 个方向的移动，每次点击在条件允许的情况下移动一格。

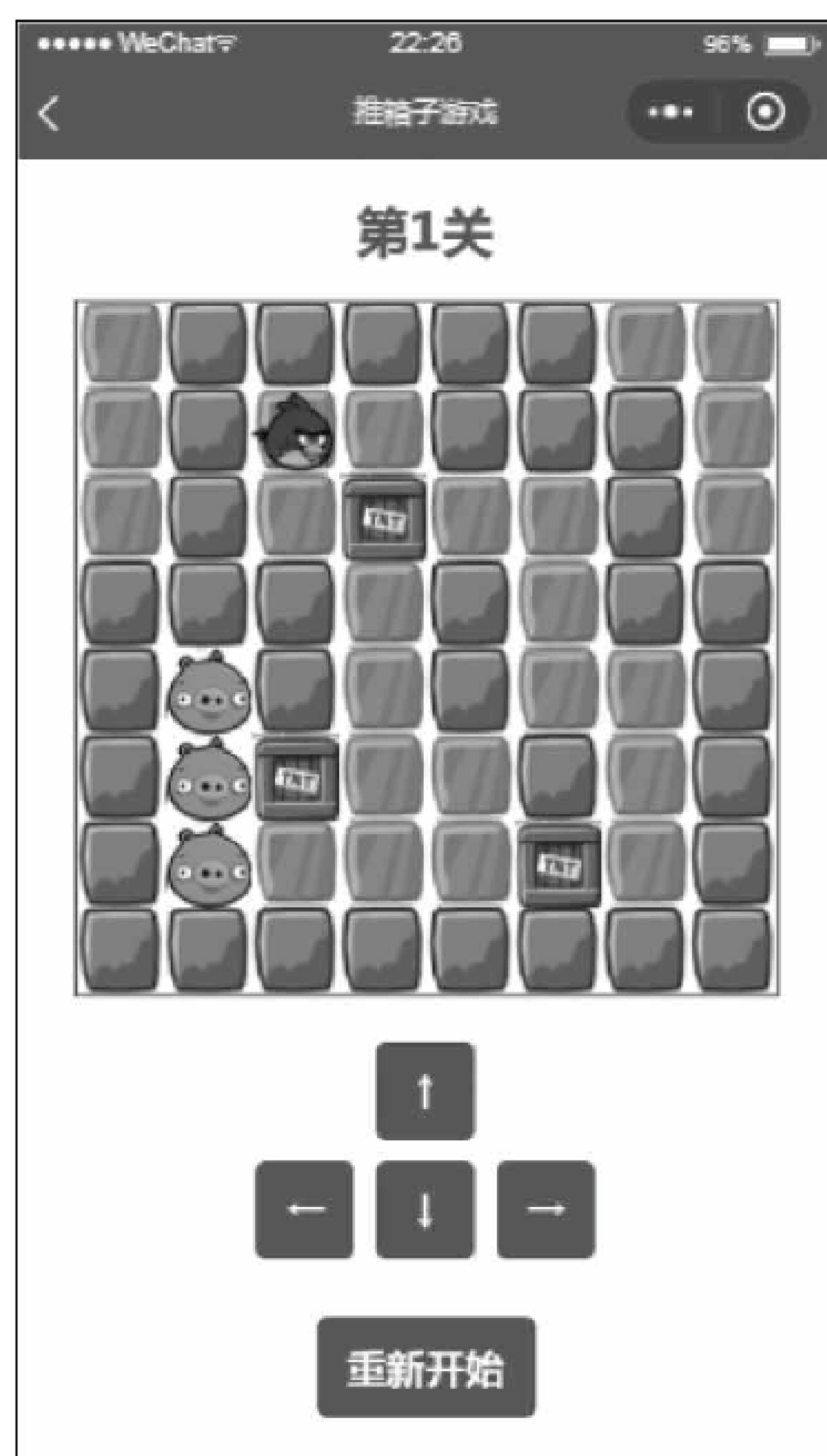
对应的 JS(pages/game/game.js)代码片段添加如下：

```
1.  Page({
2.    /**
3.     * 自定义函数 -- 方向键：上
4.     */
5.    up: function() {
6.      //不在最顶端才考虑上移
7.      if (row > 0) {
8.        //如果上方不是墙或箱子，可以移动小鸟
9.        if (map[row - 1][col] != 1 && box[row - 1][col] != 4) {
10.         //更新当前小鸟的坐标
11.         row = row - 1
12.       }
13.       //如果上方是箱子
14.       else if (box[row - 1][col] == 4) {
15.         //箱子不在最顶端才能考虑推动
16.         if (row - 1 > 0) {
17.           //如果箱子上方不是墙或箱子
18.           if (map[row - 2][col] != 1 && box[row - 2][col] != 4) {
19.             box[row - 2][col] = 4
20.             box[row - 1][col] = 0
21.             //更新当前小鸟的坐标
22.             row = row - 1
23.           }
24.         }
25.       }
26.       //重新绘制地图
27.       this.drawCanvas()
28.     }
29.   },
30.   /**
31.    * 自定义函数 -- 方向键：下
```

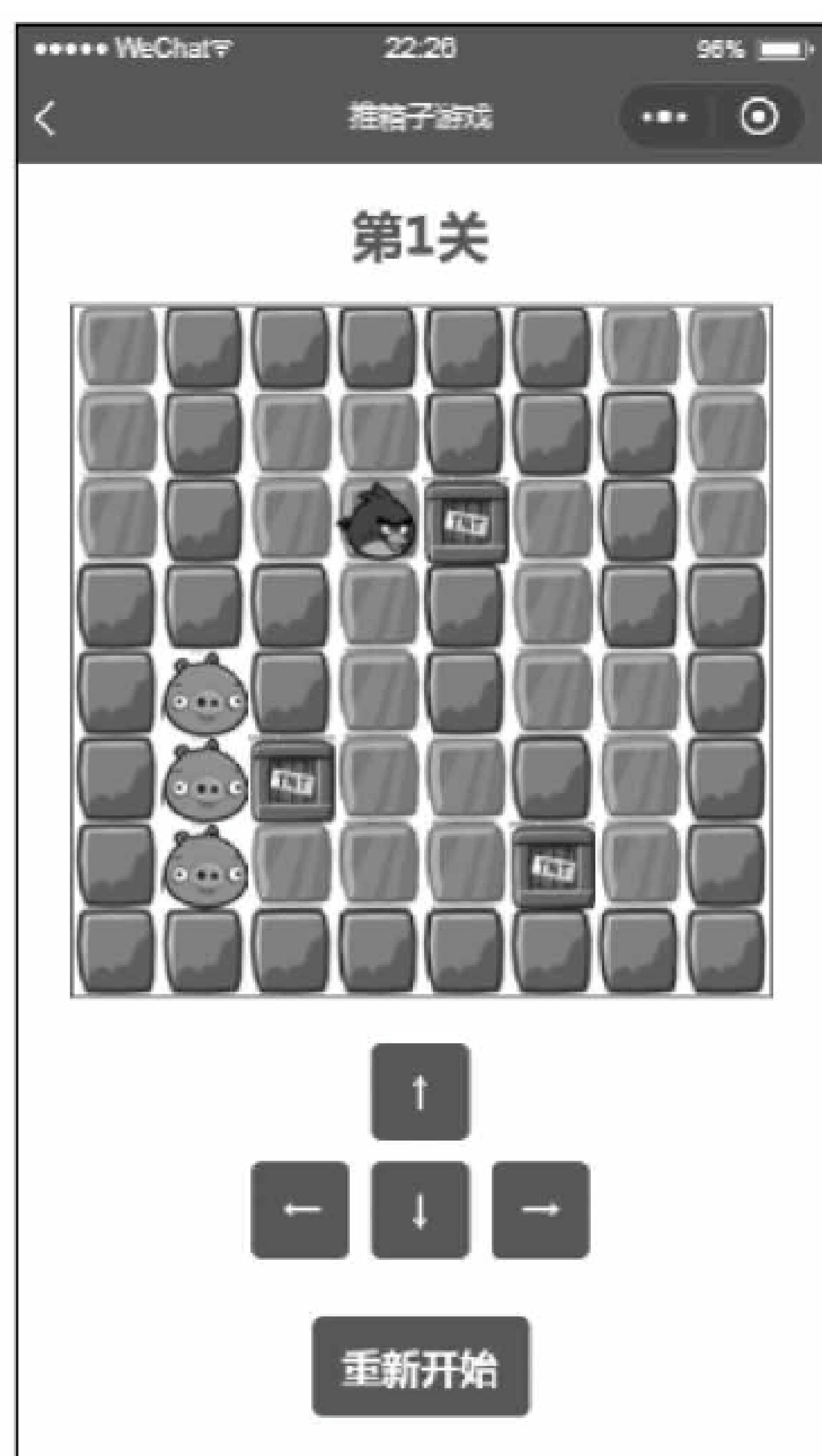
```
32.     */
33.   down: function() {
34.     //不在最底端才考虑下移
35.     if (row < 7) {
36.       //如果下方不是墙或箱子,可以移动小鸟
37.       if (map[row + 1][col] != 1 && box[row + 1][col] != 4) {
38.         //更新当前小鸟的坐标
39.         row = row + 1
40.       }
41.       //如果下方是箱子
42.       else if (box[row + 1][col] == 4) {
43.         //箱子不在最底端才能考虑推动
44.         if (row + 1 < 7) {
45.           //如果箱子下方不是墙或箱子
46.           if (map[row + 2][col] != 1 && box[row + 2][col] != 4) {
47.             box[row + 2][col] = 4
48.             box[row + 1][col] = 0
49.             //更新当前小鸟的坐标
50.             row = row + 1
51.           }
52.         }
53.       }
54.       //重新绘制地图
55.       this.drawCanvas()
56.     }
57.   },
58.   /**
59.     * 自定义函数--方向键:左
60.     */
61.   left: function() {
62.     //不在最左侧才考虑左移
63.     if (col > 0) {
64.       //如果左侧不是墙或箱子,可以移动小鸟
65.       if (map[row][col - 1] != 1 && box[row][col - 1] != 4) {
66.         //更新当前小鸟的坐标
67.         col = col - 1
68.       }
69.       //如果左侧是箱子
70.       else if (box[row][col - 1] == 4) {
71.         //箱子不在最左侧才能考虑推动
72.         if (col - 1 > 0) {
73.           //如果箱子左侧不是墙或箱子
74.           if (map[row][col - 2] != 1 && box[row][col - 2] != 4) {
75.             box[row][col - 2] = 4
76.             box[row][col - 1] = 0
77.             //更新当前小鸟的坐标
78.             col = col - 1
79.           }
80.         }
81.       }
82.       //重新绘制地图
83.       this.drawCanvas()
84.     }
85.   },
86.   /**
```

```
87.     * 自定义函数 -- 方向键：右
88.     */
89.   right: function() {
90.     //不在最右侧才考虑右移
91.     if (col < 7) {
92.       //如果右侧不是墙或箱子,可以移动小鸟
93.       if (map[row][col + 1] != 1 && box[row][col + 1] != 4) {
94.         //更新当前小鸟的坐标
95.         col = col + 1
96.       }
97.       //如果右侧是箱子
98.       else if (box[row][col + 1] == 4) {
99.         //箱子不在最右侧才能考虑推动
100.        if (col + 1 < 7) {
101.          //如果箱子右侧不是墙或箱子
102.          if (map[row][col + 2] != 1 && box[row][col + 2] != 4) {
103.            box[row][col + 2] = 4
104.            box[row][col + 1] = 0
105.            //更新当前小鸟的坐标
106.            col = col + 1
107.          }
108.        }
109.      }
110.      //重新绘制地图
111.      this.drawCanvas()
112.    }
113.  },
114.})
```

当前效果如图 13-17 所示。



(a) 向上移动效果



(b) 向右移动效果

图 13-17 游戏主角（小鸟）的移动效果

3 判断游戏成功



视频讲解

在 game.js 文件中添加自定义函数 isWin，用于判断游戏是否已经成功。

对应的 JS(pages/game/game.js)代码片段添加如下：

```
1.  Page({
2.    /**
3.     * 自定义函数 -- 判断游戏是否成功
4.     */
5.    isWin: function() {
6.      //使用双重 for 循环遍历整个数组
7.      for (var i = 0; i < 8; i++) {
8.        for (var j = 0; j < 8; j++) {
9.          //如果有箱子没在终点
10.         if (box[i][j] == 4 && map[i][j] != 3) {
11.           //返回 false,表示游戏尚未成功
12.           return false
13.         }
14.       }
15.     }
16.     //返回 true,表示游戏成功
17.     return true
18.   },
19. })
```

上述代码的判断逻辑是只要有一个箱子没有在终点位置就判断游戏尚未成功。

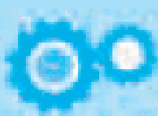
然后在 game.js 中添加自定义函数 checkWin，要求一旦游戏成功就弹出提示对话框。

对应的 JS(pages/game/game.js)代码片段修改如下：

```
1.  Page({
2.    /**
3.     * 自定义函数 -- 游戏成功处理
4.     */
5.    checkWin: function() {
6.      if (this.isWin()) {
7.        wx.showModal({
8.          title: '恭喜',
9.          content: '游戏成功!',
10.         showCancel: false
11.       })
12.     }
13.   },
14. })
```

最后在 game.js 的 4 个方向键函数中追加关于游戏成功判断的函数，这里以 up 函数为例，对应的 JS(pages/game/game.js)代码片段修改如下：

```
1.  Page({
2.    /**
3.     * 自定义函数 -- 方向键：上
```



```
4.        * /
5.      up: function() {
6.        //不在最顶端才考虑上移
7.        if (row > 0) {
8.          //如果上方不是墙或箱子,可以移动小鸟
9.          …
10.         //如果上方是箱子
11.         …
12.
13.         //重新绘制地图
14.         this.drawCanvas()
15.         //检查游戏是否成功
16.         this.checkWin()
17.       }
18.     },
19.   })
```

其他 3 个方向的函数的修改方式和 up 函数完全一致，这里不再一一赘述。游戏成功后的画面如图 13-18 所示。



视频讲解



图 13-18 游戏成功提示画面

4 重新开始游戏

修改 game.wxml 代码，为“重新开始”按钮追加自定义函数的点击事件。

WXML(pages/game/game.wxml)代码片段修改如下：

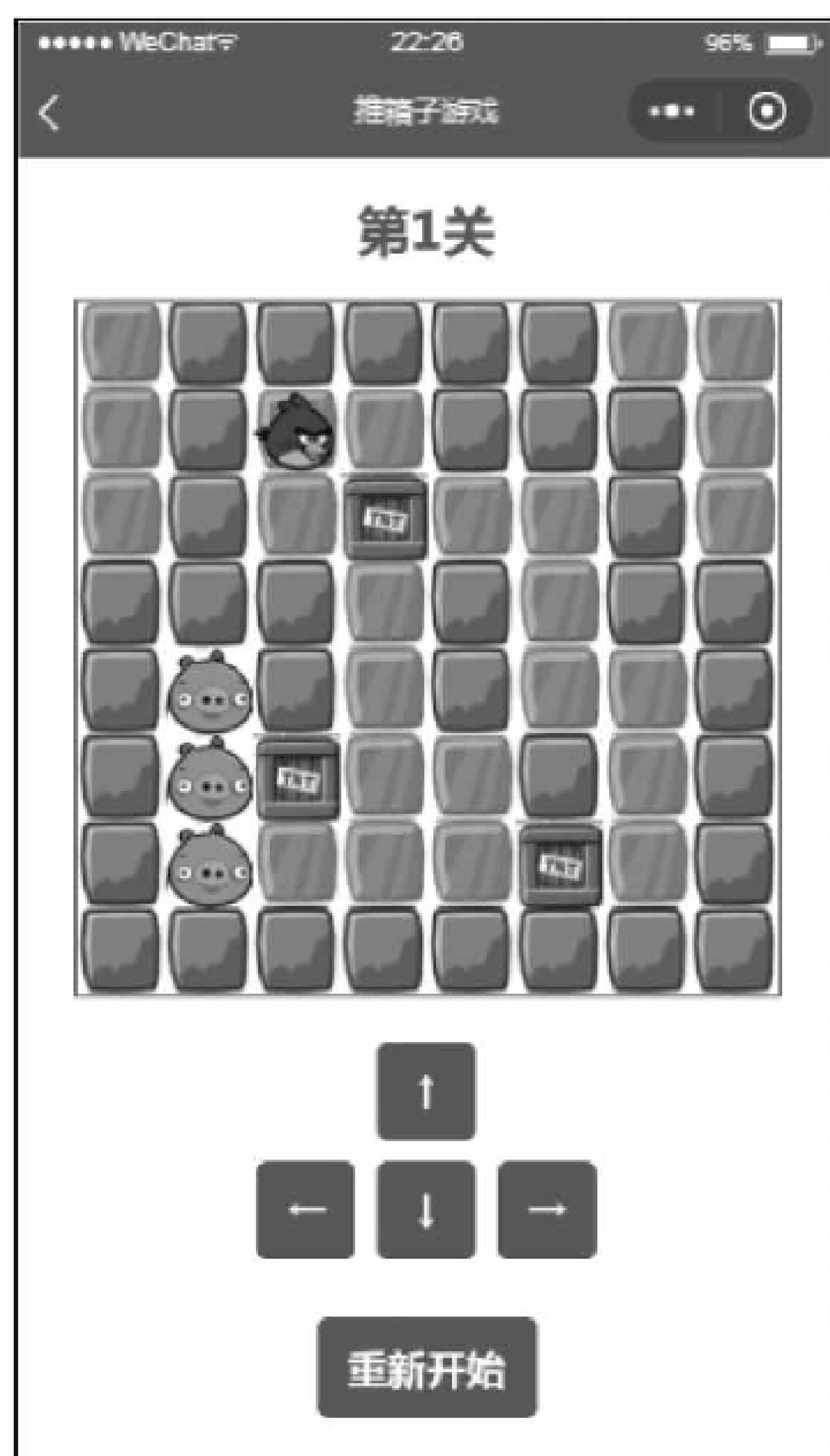
```
1.  < view class = 'container'>
2.     …
3.
4.    <!-- “重新开始”按钮 -->
5.    < button type = 'warn' bindtap = 'restartGame'>重新开始</button >
6.  </view >
```

在 game.js 文件中添加 restartGame 函数，用于重新开始游戏。

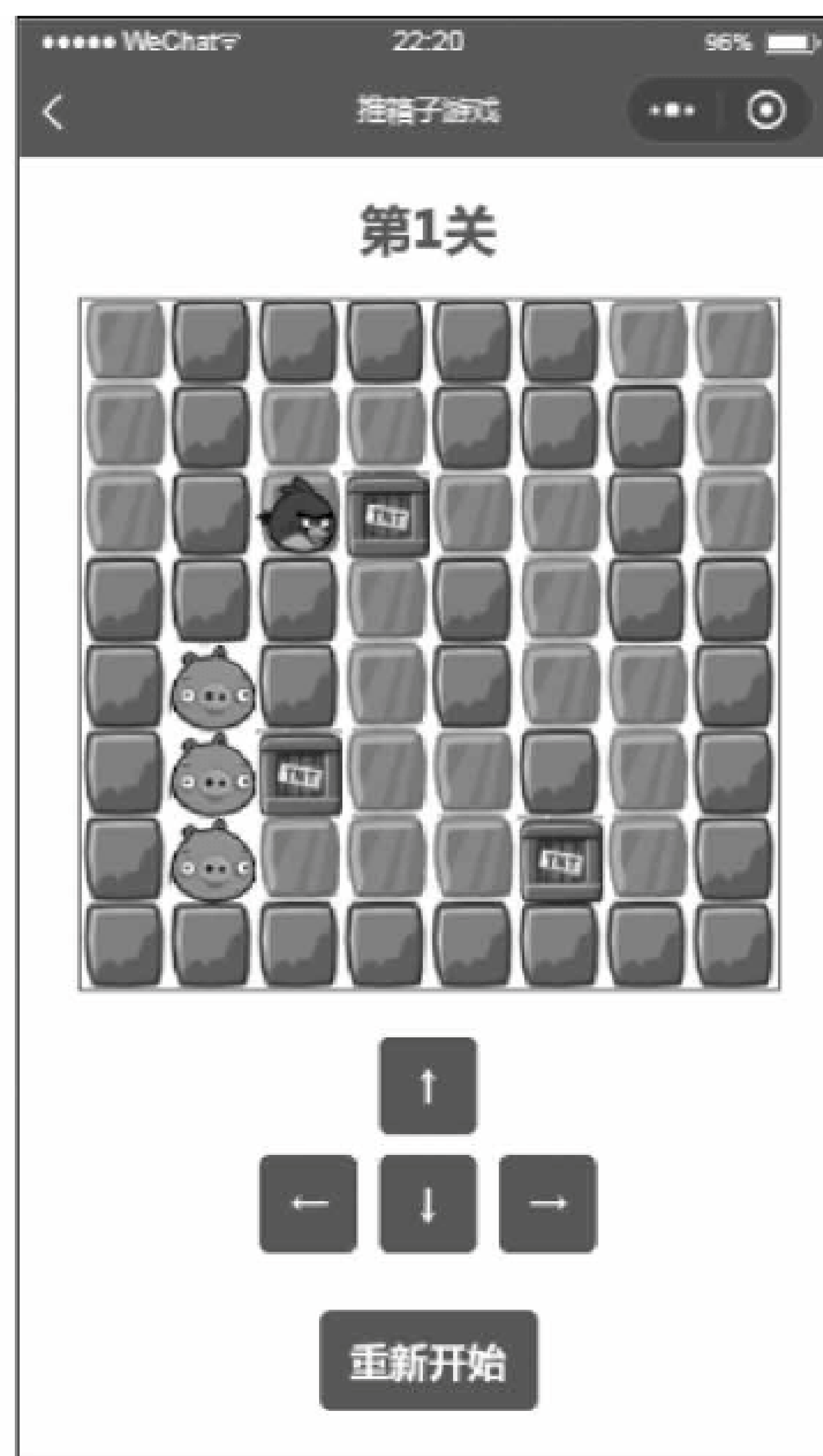
对应的 JS(pages/game/game.js)代码片段添加如下：

```
1.  Page({
2.    / **
3.      * 自定义函数 -- 重新开始游戏
4.      * /
5.    restartGame: function() {
6.      //初始化地图数据
7.      this.initMap(this.data.level - 1)
8.      //绘制画布内容
9.      this.drawCanvas()
10.   },
11. })
```

此时页面效果如图 13-19 所示。

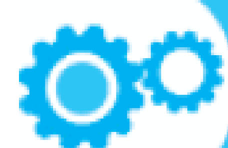


(a) 向上移动效果



(b) 重新开始游戏

图 13-19 点击“重新开始”按钮效果



13.6 完整代码展示



13.6.1 应用文件代码展示

1 app.json 代码展示

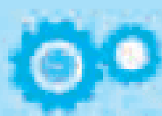
app.json 文件的完整代码如下：

```
1.  {
2.    "pages": [
3.      "pages/index/index",
4.      "pages/game/game"
5.    ],
6.    "window": {
7.      "navigationBarBackgroundColor": "＃E64340",
8.      "navigationBarTitleText": "推箱子游戏"
9.    }
10. }
```

2 app.wxss 代码展示

app.wxss 文件的完整代码如下：

```
1.  /＊页面容器样式＊/
2.  .container {
3.    height: 100vh;
4.    color: ＃E64340;
5.    font－weight: bold;
6.    display: flex;
```

```
7.      flex-direction: column;
8.      align-items: center;
9.      justify-content: space-evenly;
10.  }
11.  /* 顶端标题样式 */
12.  .title {
13.      font-size: 18pt;
14.  }
```

13.6.2 公共函数文件代码展示

JS 文件(utils/data.js)的完整代码如下：

```
1.   //==================================================
2.   //地图数据 map1～map4
3.   //地图数据：1 为墙、2 为路、3 为终点、4 为箱子、5 为人物、0 为墙的外围
4.   //==================================================
5.   //关卡 1
6.   var map1 = [
7.     [0, 1, 1, 1, 1, 1, 0, 0],
8.     [0, 1, 2, 2, 1, 1, 1, 0],
9.     [0, 1, 5, 4, 2, 2, 1, 0],
10.    [1, 1, 1, 2, 1, 2, 1, 1],
11.    [1, 3, 1, 2, 1, 2, 2, 1],
12.    [1, 3, 4, 2, 2, 1, 2, 1],
13.    [1, 3, 2, 2, 2, 4, 2, 1],
14.    [1, 1, 1, 1, 1, 1, 1, 1]
15.  ]
16.  //关卡 2
17.  var map2 = [
18.    [0, 0, 1, 1, 1, 0, 0, 0],
19.    [0, 0, 1, 3, 1, 0, 0, 0],
20.    [0, 0, 1, 2, 1, 1, 1, 1],
21.    [1, 1, 1, 4, 2, 4, 3, 1],
22.    [1, 3, 2, 4, 5, 1, 1, 1],
23.    [1, 1, 1, 1, 4, 1, 0, 0],
24.    [0, 0, 0, 1, 3, 1, 0, 0],
25.    [0, 0, 0, 1, 1, 1, 0, 0]
26.
27.  ]
28.  //关卡 3
29.  var map3 = [
30.    [0, 0, 1, 1, 1, 1, 0, 0],
31.    [0, 0, 1, 3, 3, 1, 0, 0],
32.    [0, 1, 1, 2, 3, 1, 1, 0],
33.    [0, 1, 2, 2, 4, 3, 1, 0],
34.    [1, 1, 2, 2, 5, 4, 1, 1],
35.    [1, 2, 2, 1, 4, 4, 2, 1],
36.    [1, 2, 2, 2, 2, 2, 2, 1],
37.    [1, 1, 1, 1, 1, 1, 1, 1]
38.  ]
39.  //关卡 4
40.  var map4 = [
41.    [0, 1, 1, 1, 1, 1, 1, 0],
42.    [0, 1, 3, 2, 3, 3, 1, 0],
```

```
43.     [0, 1, 3, 2, 4, 3, 1, 0],
44.     [1, 1, 1, 2, 2, 4, 1, 1],
45.     [1, 2, 4, 2, 2, 4, 2, 1],
46.     [1, 2, 1, 4, 1, 1, 2, 1],
47.     [1, 2, 2, 2, 5, 2, 2, 1],
48.     [1, 1, 1, 1, 1, 1, 1, 1]
49.  ]
50.
51.  module.exports = {
52.     maps: [map1, map2, map3, map4]
53.  }
```

13.6.3 页面文件代码展示

1 首页代码展示

WXML 文件(pages/index/index.wxml)的完整代码如下:

```
1.  < view class = 'container'>
2.     <!-- 标题 -->
3.     < view class = 'title'>游戏选关</view >
4.
5.     <!-- 关卡列表 -->
6.     < view class = 'levelBox'>
7.       < view class = 'box' wx:for = '{{levels}}' wx:key = 'levels{{index}}' bindtap = 'chooseLevel'
data - level = '{{index}}'>
8.          < image src = '/images/{{item}}'></image >
9.          < text >第{{index + 1}}关</text >
10.      </view >
11.    </view >
12.
13. </view >
```

WXSS 文件(pages/index/index.wxss)的完整代码如下:

```
1.  /* 关卡列表区域 */
2.  .levelBox {
3.     width: 100 % ;
4.  }
5.
6.  /* 单个关卡区域 */
7.  .box {
8.     width: 50 % ;
9.     float: left;
10.    margin: 20rpx 0;
11.    display: flex;
12.    flex - direction: column;
13.    align - items: center;
14. }
15.
16. /* 选关图片 */
17. image {
18.    width: 300rpx;
19.    height: 300rpx;
20. }
```

JS 文件(pages/index/index.js)的完整代码如下：

```
1.  Page({
2.    /**
3.     * 页面的初始数据
4.     */
5.    data: {
6.      levels: [
7.        'level01.png',
8.        'level02.png',
9.        'level03.png',
10.       'level04.png'
11.     ]
12.   },
13.   /**
14.    * 自定义函数 -- 游戏选关
15.    */
16.   chooseLevel: function(e) {
17.     let level = e.currentTarget.dataset.level
18.     wx.navigateTo({
19.       url: '../game/game?level=' + level
20.     })
21.   },
22. })
```

2 游戏页代码展示

WXML 文件(pages/game/game.wxml)的完整代码如下：

```
1.  <view class='container'>
2.    <!-- 关卡提示 -->
3.    <view class='title'>第{{level}}关</view>
4.
5.    <!-- 游戏画布 -->
6.    <canvas canvas-id='myCanvas'></canvas>
7.
8.    <!-- 方向键 -->
9.    <view class='btnBox'>
10.     <button type='warn' bindtap='up'>↑</button>
11.     <view>
12.       <button type='warn' bindtap='left'>←</button>
13.       <button type='warn' bindtap='down'>↓</button>
14.       <button type='warn' bindtap='right'>→</button>
15.     </view>
16.   </view>
17.
18.   <!-- "重新开始"按钮 -->
19.   <button type='warn' bindtap='restartGame'>重新开始</button>
20. </view>
```

WXSS 文件(pages/game/game.wxss)的完整代码如下：

```
1.  /*游戏画布样式*/
2.  canvas {
3.    border: 1rpx solid;
4.    width: 320px;
5.    height: 320px;
```

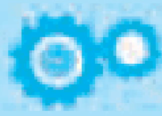
```
6.  }
7.
8.  /* 方向键按钮整体区域 */
9.  .btnBox {
10.   display: flex;
11.   flex-direction: column;
12.   align-items: center;
13. }
14.
15. /* 方向键按钮第二行 */
16. .btnBox view {
17.   display: flex;
18.   flex-direction: row;
19. }
20.
21. /* 所有方向键按钮 */
22. .btnBox button {
23.   width: 90rpx;
24.   height: 90rpx;
25. }
26.
27. /* 所有按钮样式 */
28. button {
29.   margin: 10rpx;
30. }
```

JS 文件(pages/game/game.js)的完整代码如下：

```
1.  var data = require('../../utils/data.js')
2.  //地图图层数据
3.  var map = [
4.    [0, 0, 0, 0, 0, 0, 0, 0],
5.    [0, 0, 0, 0, 0, 0, 0, 0],
6.    [0, 0, 0, 0, 0, 0, 0, 0],
7.    [0, 0, 0, 0, 0, 0, 0, 0],
8.    [0, 0, 0, 0, 0, 0, 0, 0],
9.    [0, 0, 0, 0, 0, 0, 0, 0],
10.   [0, 0, 0, 0, 0, 0, 0, 0],
11.   [0, 0, 0, 0, 0, 0, 0, 0]
12. ]
13. //箱子图层数据
14. var box = [
15.   [0, 0, 0, 0, 0, 0, 0, 0],
16.   [0, 0, 0, 0, 0, 0, 0, 0],
17.   [0, 0, 0, 0, 0, 0, 0, 0],
18.   [0, 0, 0, 0, 0, 0, 0, 0],
19.   [0, 0, 0, 0, 0, 0, 0, 0],
20.   [0, 0, 0, 0, 0, 0, 0, 0],
21.   [0, 0, 0, 0, 0, 0, 0, 0],
22.   [0, 0, 0, 0, 0, 0, 0, 0]
23. ]
24. //方块的宽度
25. var w = 40
26. //初始化小鸟的行与列
27. var row = 0
28. var col = 0
```

```
29.
30. Page({
31.
32.   /**
33.    * 页面的初始数据
34.    */
35.   data: {
36.     level: 1
37.   },
38.
39.   /**
40.    * 自定义函数--初始化地图数据
41.    */
42.   initMap: function(level) {
43.     //读取原始的游戏地图数据
44.     let mapData = data.maps[level]
45.     //使用双重 for 循环记录地图数据
46.     for (var i = 0; i < 8; i++) {
47.       for (var j = 0; j < 8; j++) {
48.         box[i][j] = 0
49.         map[i][j] = mapData[i][j]
50.
51.         if (mapData[i][j] == 4) {
52.           box[i][j] = 4
53.           map[i][j] = 2
54.         } else if (mapData[i][j] == 5) {
55.           map[i][j] = 2
56.           //记录小鸟的当前行和列
57.           row = i
58.           col = j
59.         }
60.       }
61.     }
62.   },
63.   /**
64.    * 自定义函数--绘制地图
65.    */
66.   drawCanvas: function() {
67.     let ctx = this.ctx
68.     //清空画布
69.     ctx.clearRect(0, 0, 320, 320)
70.     //使用双重 for 循环绘制 8×8 的地图
71.     for (var i = 0; i < 8; i++) {
72.       for (var j = 0; j < 8; j++) {
73.         //默认是道路
74.         let img = 'ice'
75.         if (map[i][j] == 1) {
76.           img = 'stone'
77.         } else if (map[i][j] == 3) {
78.           img = 'pig'
79.         }
80.
81.         //绘制地图
82.         ctx.drawImage('/images/icons/' + img + '.png', j * w, i * w, w, w)
83.
```

```
84.         if (box[i][j] == 4) {
85.           //叠加绘制箱子
86.           ctx.drawImage('/images/icons/box.png', j * w, i * w, w, w)
87.         }
88.       }
89.     }
90.
91.     //叠加绘制小鸟
92.     ctx.drawImage('/images/icons/bird.png', col * w, row * w, w, w)
93.
94.     ctx.draw()
95.   },
96.
97.   /**
98.    * 自定义函数 -- 方向键：上
99.    */
100.  up: function() {
101.    //不在最顶端才考虑上移
102.    if (row > 0) {
103.      //如果上方不是墙或箱子,可以移动小鸟
104.      if (map[row - 1][col] != 1 && box[row - 1][col] != 4) {
105.        //更新当前小鸟的坐标
106.        row = row - 1
107.      }
108.      //如果上方是箱子
109.      else if (box[row - 1][col] == 4) {
110.        //箱子不在最顶端才能考虑推动
111.        if (row - 1 > 0) {
112.          //如果箱子上方不是墙或箱子
113.          if (map[row - 2][col] != 1 && box[row - 2][col] != 4) {
114.            box[row - 2][col] = 4
115.            box[row - 1][col] = 0
116.            //更新当前小鸟的坐标
117.            row = row - 1
118.          }
119.        }
120.      }
121.      //重新绘制地图
122.      this.drawCanvas()
123.      //检查游戏是否成功
124.      this.checkWin()
125.    }
126.  },
127.  /**
128.   * 自定义函数 -- 方向键：下
129.   */
130.  down: function() {
131.    //不在最底端才考虑下移
132.    if (row < 7) {
133.      //如果下方不是墙或箱子,可以移动小鸟
134.      if (map[row + 1][col] != 1 && box[row + 1][col] != 4) {
135.        //更新当前小鸟的坐标
136.        row = row + 1
137.      }
138.      //如果下方是箱子
```

```
139.        else if (box[row + 1][col] == 4) {
140.          //箱子不在最底端才能考虑推动
141.          if (row + 1 < 7) {
142.            //如果箱子下方不是墙或箱子
143.            if (map[row + 2][col] != 1 && box[row + 2][col] != 4) {
144.              box[row + 2][col] = 4
145.              box[row + 1][col] = 0
146.              //更新当前小鸟的坐标
147.              row = row + 1
148.            }
149.          }
150.        }
151.        //重新绘制地图
152.        this.drawCanvas()
153.        //检查游戏是否成功
154.        this.checkWin()
155.      }
156.    },
157.    /**
158.     * 自定义函数 -- 方向键: 左
159.     */
160.    left: function() {
161.      //不在最左侧才考虑左移
162.      if (col > 0) {
163.        //如果左侧不是墙或箱子,可以移动小鸟
164.        if (map[row][col - 1] != 1 && box[row][col - 1] != 4) {
165.          //更新当前小鸟的坐标
166.          col = col - 1
167.        }
168.        //如果左侧是箱子
169.        else if (box[row][col - 1] == 4) {
170.          //箱子不在最左侧才能考虑推动
171.          if (col - 1 > 0) {
172.            //如果箱子左侧不是墙或箱子
173.            if (map[row][col - 2] != 1 && box[row][col - 2] != 4) {
174.              box[row][col - 2] = 4
175.              box[row][col - 1] = 0
176.              //更新当前小鸟的坐标
177.              col = col - 1
178.            }
179.          }
180.        }
181.        //重新绘制地图
182.        this.drawCanvas()
183.        //检查游戏是否成功
184.        this.checkWin()
185.      }
186.
187.
188.    },
189.    /**
190.     * 自定义函数 -- 方向键: 右
191.     */
192.    right: function() {
193.      //不在最右侧才考虑右移
```

```
194.     if (col < 7) {
195.       //如果右侧不是墙或箱子,可以移动小鸟
196.       if (map[row][col + 1] != 1 && box[row][col + 1] != 4) {
197.         //更新当前小鸟的坐标
198.         col = col + 1
199.       }
200.       //如果右侧是箱子
201.       else if (box[row][col + 1] == 4) {
202.         //箱子不在最右侧才能考虑推动
203.         if (col + 1 < 7) {
204.           //如果箱子右侧不是墙或箱子
205.           if (map[row][col + 2] != 1 && box[row][col + 2] != 4) {
206.             box[row][col + 2] = 4
207.             box[row][col + 1] = 0
208.             //更新当前小鸟的坐标
209.             col = col + 1
210.           }
211.         }
212.       }
213.       //重新绘制地图
214.       this.drawCanvas()
215.       //检查游戏是否成功
216.       this.checkWin()
217.     }
218.   },
219.
220.
221.
222.   /**
223.    * 自定义函数 -- 判断游戏是否成功
224.    */
225.   isWin: function() {
226.     //使用双重 for 循环遍历整个数组
227.     for (var i = 0; i < 8; i++) {
228.       for (var j = 0; j < 8; j++) {
229.         //如果有箱子没在终点
230.         if (box[i][j] == 4 && map[i][j] != 3) {
231.           //返回 false,表示游戏尚未成功
232.           return false
233.         }
234.       }
235.     }
236.     //返回 true,表示游戏成功
237.     return true
238.   },
239.
240.   /**
241.    * 自定义函数 -- 游戏成功处理
242.    */
243.   checkWin: function() {
244.     if (this.isWin()) {
245.       wx.showModal({
246.         title: '恭喜',
247.         content: '游戏成功!',
248.         showCancel: false
```

```
249.       })
250.     }
251.   },
252.
253.   /**
254.    * 自定义函数--重新开始游戏
255.    */
256.   restartGame: function() {
257.     //初始化地图数据
258.     this.initMap(this.data.level - 1)
259.     //绘制画布内容
260.     this.drawCanvas()
261.   },
262.
263.   /**
264.    * 生命周期函数--监听页面加载
265.    */
266.   onLoad: function(options) {
267.     //获取关卡
268.     let level = options.level
269.     //更新页面关卡标题
270.     this.setData({
271.       level: parseInt(level) + 1
272.     })
273.     //创建画布上下文
274.     this.ctx = wx.createCanvasContext('myCanvas')
275.     //初始化地图数据
276.     this.initMap(level)
277.     //绘制画布内容
278.     this.drawCanvas()
279.   },
280.})
```

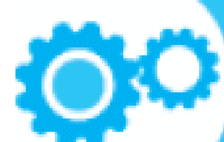


第14章 Chapter 14 小程序游戏·贪吃蛇游戏

在学习了<canvas>(画布)组件和小程序界面API中绘图的相关用法以后，读者不妨尝试制作简易的贪吃蛇小游戏。

本章学习目标

- 综合应用所学知识创建完整的贪吃蛇游戏项目；
- 熟练掌握<canvas>(画布)组件和绘图API。



14.1 需求分析





视频讲解

本项目一共需要两个页面，即首页和游戏页面，其中首页用于呈现关卡菜单，点击对应难度的关卡后进入游戏画面。

14.1.1 首页功能需求

首页功能需求如下：

(1) 首页需要包含标题和关卡列表。

(2) 关卡列表包含两种游戏模式，即简单模式和困难模式，主要区别在于贪吃蛇移动速度的快慢。

(3) 点击关卡图片可以打开对应的游戏画面。

14.1.2 游戏页功能需求

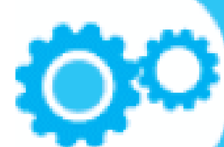
游戏页功能需求如下：

(1) 游戏页面需要显示当前得分、游戏画面、方向键和“重新开始”按钮。

(2) 点击方向键可以使贪吃蛇上、下、左、右转方向前进和吃食物。

(3) 游戏画面由16×16格的小方块组成，主要用于显示贪吃蛇和食物。

(4) 点击“重新开始”按钮可以重置全部游戏数据并重新开始游戏。



14.2 项目创建





视频讲解

本项目创建选择空白文件夹snakeDemo，效果如图14-1所示。

单击“新建”按钮完成项目创建，然后准备手动修改页面配置文件。

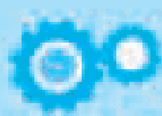
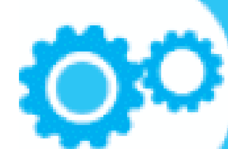




图 14-1　小程序项目填写效果示意图



14.3　页面配置



14.3.1　创建页面文件



视频讲解

项目创建完毕后，在根目录中会生成文件夹 pages 用于存放页面文件。一般来说首页默认命名为 index，表示小程序运行的第一个页面；其他页面名称可以自定义。本项目有两个页面文件，需要创建 index（首页页面）和 game（游戏页面）。

具体操作如下：

（1）将 app.json 文件内 pages 属性中的"pages/logs/logs"改成"pages/game/game"；

（2）按快捷键 Ctrl+S 保存修改后会在 pages 文件夹下自动生成 game 目录。

14.3.2　删除和修改文件

具体操作如下：

（1）删除 utils 文件夹及其内部所有内容。

（2）删除 pages 文件夹下的 logs 目录及其内部所有内容。

（3）删除 index.wxml 和 index.wxss 中的全部代码。

（4）删除 index.js 中的全部代码，并且输入关键词 page 找到第二个选项按回车键让其自动补全函数（如图 14-2 所示）。

（5）删除 app.wxss 中的全部代码。

（6）删除 app.js 中的全部代码，并且输入关键词 app 找到第一个选项按回车键让其自动补全函数（如图 14-3 所示）。

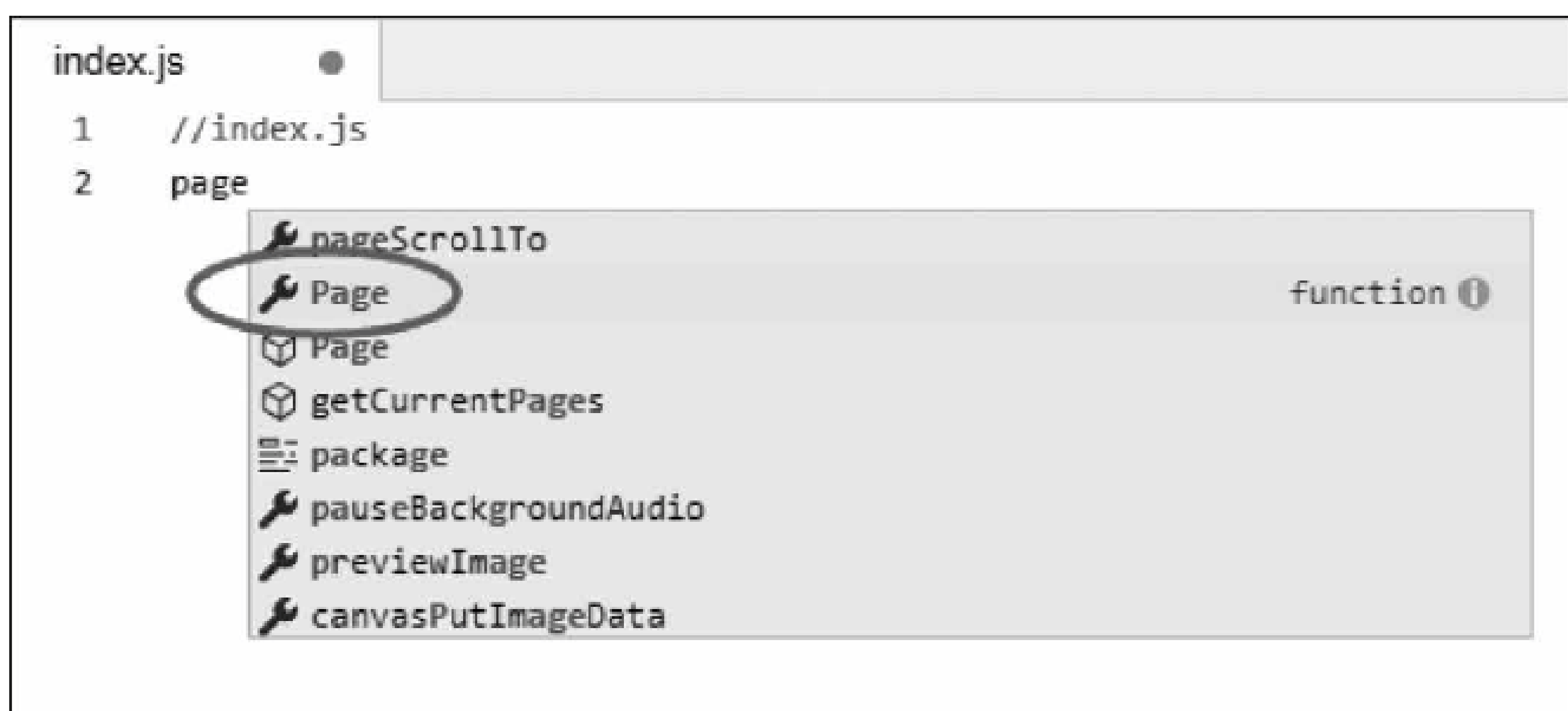


图 14-2　输入关键词创建 Page 函数

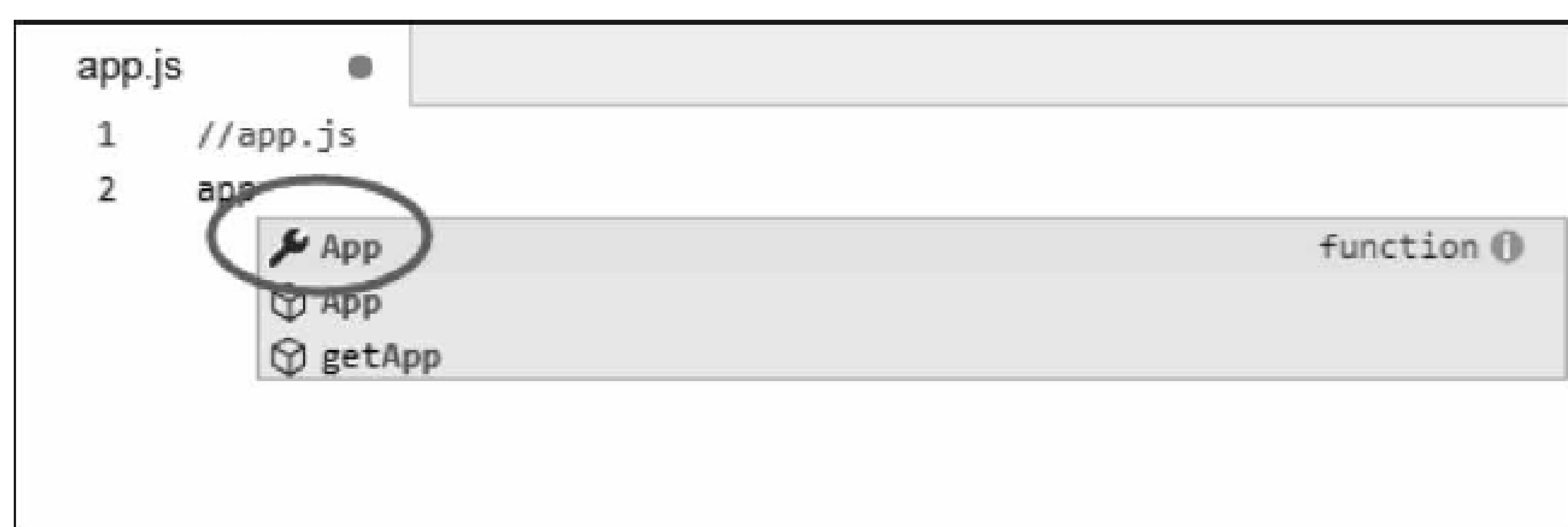


图 14-3　输入关键词创建 App 函数

14.3.3　创建其他文件

接下来创建其他自定义文件，本项目还需要一个 images 文件夹，用于存放图片素材。单击目录结构左上角的＋号创建文件夹，并命名为 images。

本项目将在首页中用到两幅关卡图片，图片素材如图 14-4 所示。

右击目录结构中的 images 文件夹，选择“硬盘打开”，将图片复制、粘贴进去。

全部完成后的目录结构如图 14-5 所示。

此时文件配置就全部完成，14.4 节将正式进行页面布局和样式设计。



(a) snake01.png

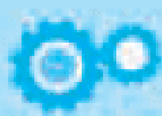


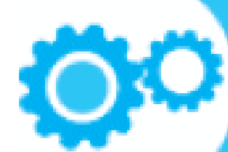
(b) snake02.png

图 14-4　关卡图片素材展示



图 14-5　全部文件创建完成



14.4 视图设计



14.4.1 导航栏设计



视频讲解

小程序默认导航栏是黑底白字的效果，可以通过在 app.json 中对 window 属性进行重新配置来自定义导航栏效果。更改后的 app.json 文件代码如下：

```
1.  {
2.    "pages": [代码略],
3.    "window":{
4.      "navigationBarBackgroundColor": "#E64340",
5.      "navigationBarTitleText": "贪吃蛇小游戏"
6.    }
7.  }
```

上述代码可以更改导航栏背景色为珊瑚红色、字体为白色，效果如图 14-6 所示。

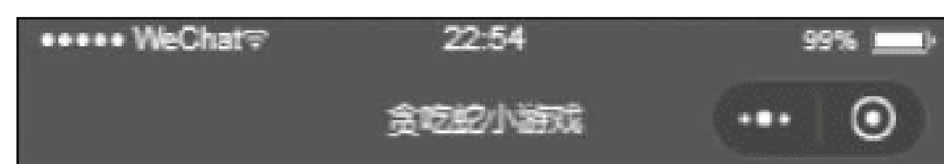


图 14-6　自定义导航栏效果

14.4.2 页面设计



视频讲解

1 公共样式设计

首先在 app.wxss 中设置页面容器和顶端标题的公共样式，代码如下：

```
1.  /* 页面容器样式 */
2.  .container {
3.    height: 100vh;
4.    display: flex;
5.    flex-direction: column;
6.    align-items: center;
7.    justify-content: space-evenly;
8.  }
```

2 首页设计



视频讲解

首页主要包含两个关卡选项，页面设计如图 14-7 所示。

计划使用如下组件。

- 整体：<view>容器；
- 关卡列表：<image>(图片)组件。

WXML(pages/index/index.wxml)代码如下：

```
1.  <view class="container">
2.    <image src='/images/snake01.png'></image>
3.    <image src='/images/snake02.png'></image>
4.  </view>
```

相关 WXSS(pages/index/index.wxss)代码片段如下：

```
1.  /* 关卡图片 */
2.  image{
3.    width: 400rpx;
```

```
4.     height: 400rpx;
5.  }
```

当前效果如图 14-8 所示。

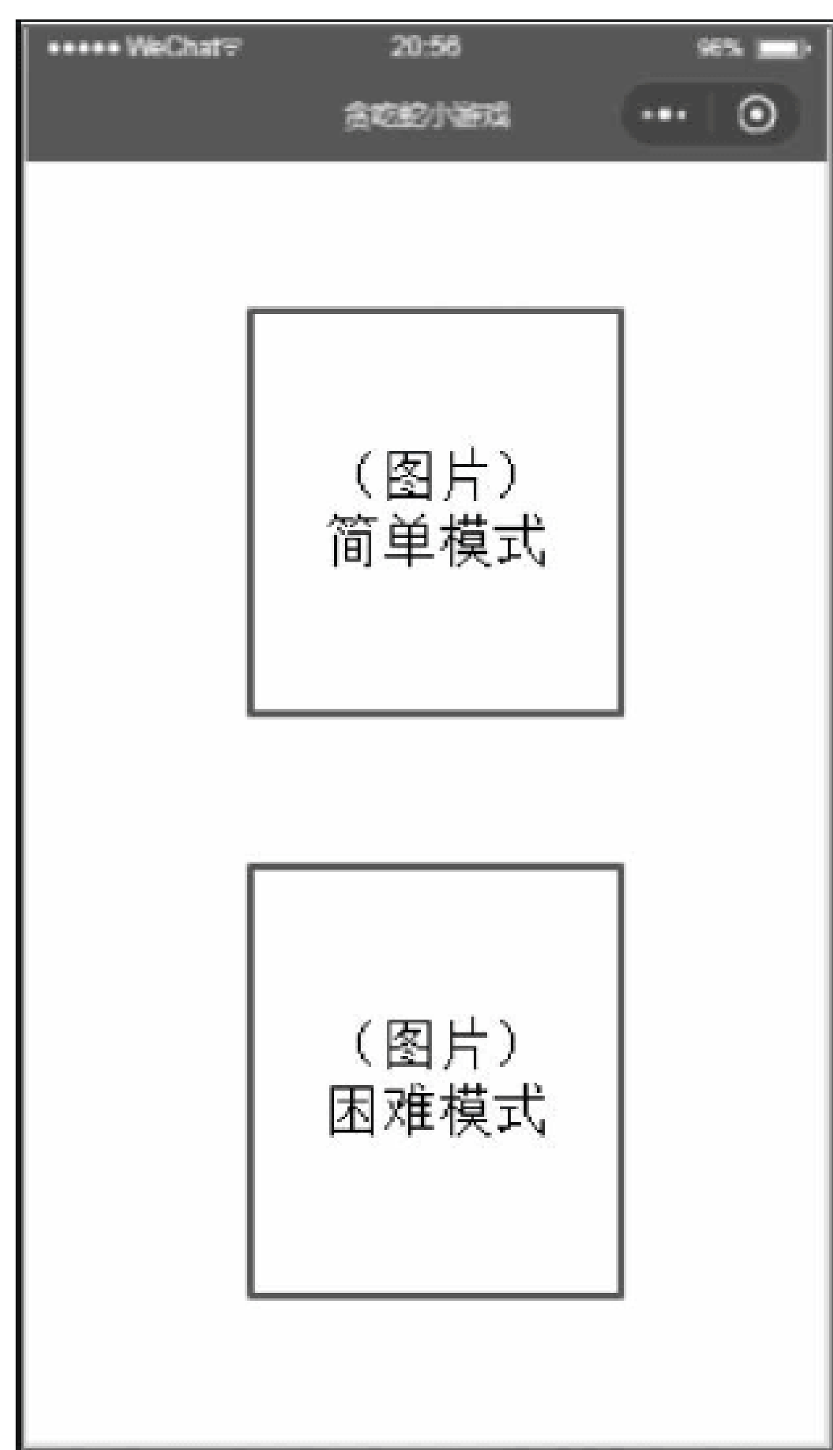


图 14-7 首页设计图



图 14-8 首页效果图

由图可见，此时可以显示首页的两种游戏模式。由于尚未编写图片的点击事件，所以暂时无法跳转游戏页面，仅供作为样式参考。


视频讲解

3 游戏页面设计

游戏页面需要用户点击首页的游戏模式，然后在新窗口中打开该页面。游戏页面包括当前分数、游戏区域、方向键和“重新开始”按钮，页面设计如图 14-9 所示。



图 14-9 游戏页面设计图

由于暂时没有做点击跳转的逻辑设计，所以可以在开发工具顶端选择“普通编译”下的“添加编译模式”，并携带临时测试参数 time=500，表示游戏刷新频率为每隔 500ms（即 0.5s）刷新一次，如图 14-10 所示。

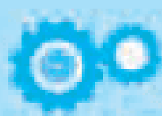
此时预览就可以直接显示 game 页面了，设计完毕后再改回“普通编译”模式即可重新显示首页。

计划使用如下组件。

- <view>：整体容器和顶端标题；
- <canvas>：游戏画布；
- <button>：4 个方向键和 1 个“重新开始”按钮。

WXML（pages/game/game.wxml）代码如下：

```
1.  <view class = 'container'>
2.      <!-- 关卡提示 -->
```



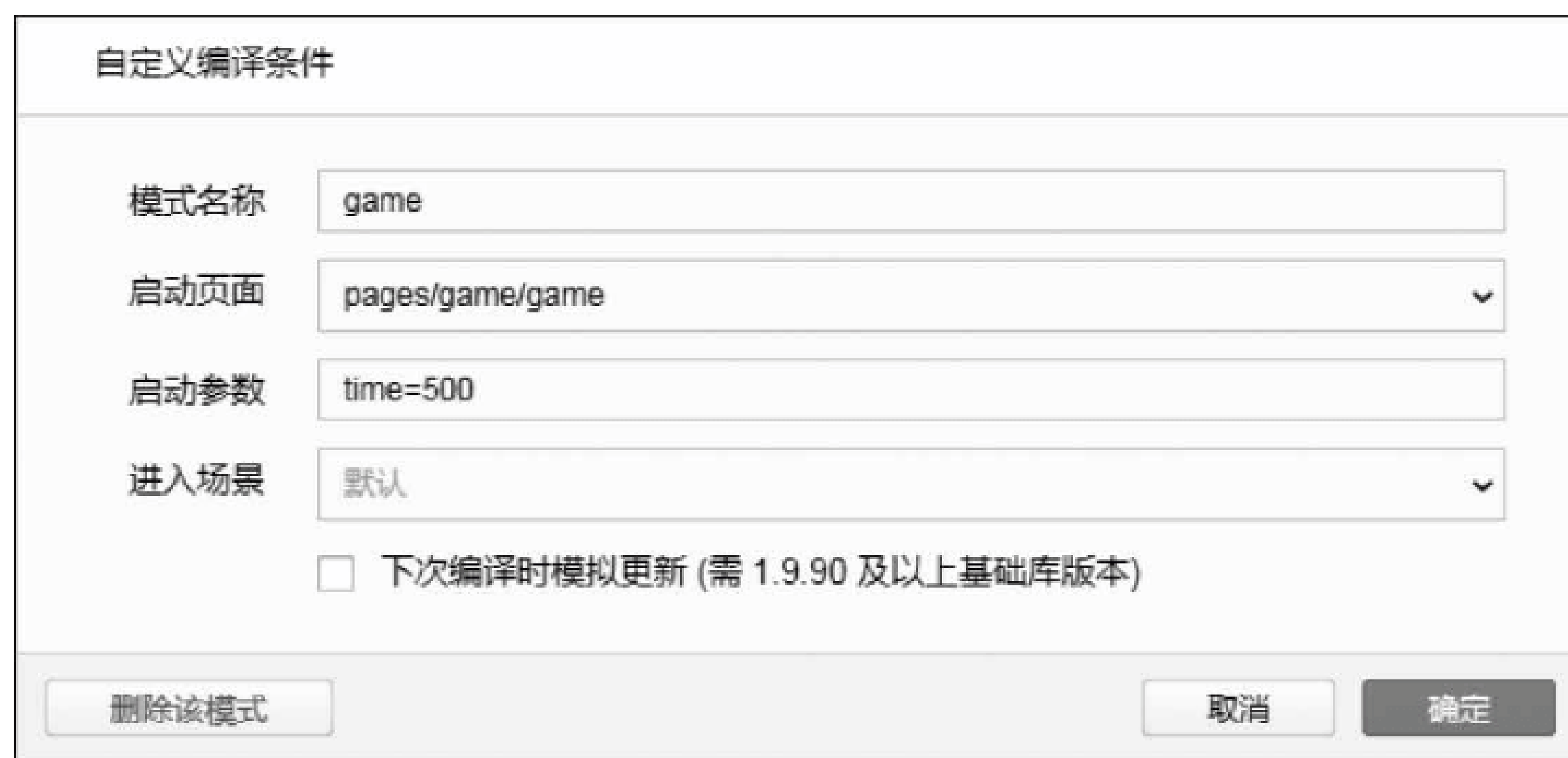


图 14-10 添加 game 页面的编译模式

```
3.    <view>当前分数：0</view>
4.
5.    <!-- 游戏画布 -->
6.    <canvas canvas-id='myCanvas'></canvas>
7.
8.    <!-- 方向键 -->
9.    <view class='btnBox'>
10.     <button>↑</button>
11.     <view>
12.       <button>←</button>
13.       <button>↓</button>
14.       <button>→</button>
15.     </view>
16.   </view>
17.
18.   <!-- "重新开始"按钮 -->
19.   <button>重新开始</button>
20. </view>
```

WXSS(pages/game/game.wxss)代码如下：

```
1.  /* 游戏画布样式 */
2.  canvas {
3.    border: 1rpx solid #E64340;
4.    width: 320px;
5.    height: 320px;
6.  }
7.
8.  /* 方向键按钮整体区域 */
9.  .btnBox {
10.   display: flex;
11.   flex-direction: column;
12.   align-items: center;
13. }
14.
15. /* 方向键按钮第二行 */
16. .btnBox view {
17.   display: flex;
18.   flex-direction: row;
19. }
20.
21. /* 所有方向键按钮 */
22. .btnBox button {
```

```
23.    width: 90rpx;
24.    height: 90rpx;
25. }
26.
27. /* 所有按钮样式 */
28. button {
29.    margin: 10rpx;
30.    background-color: #E64340;
31.    color: white;
32. }
```

当前效果如图 14-11 所示。

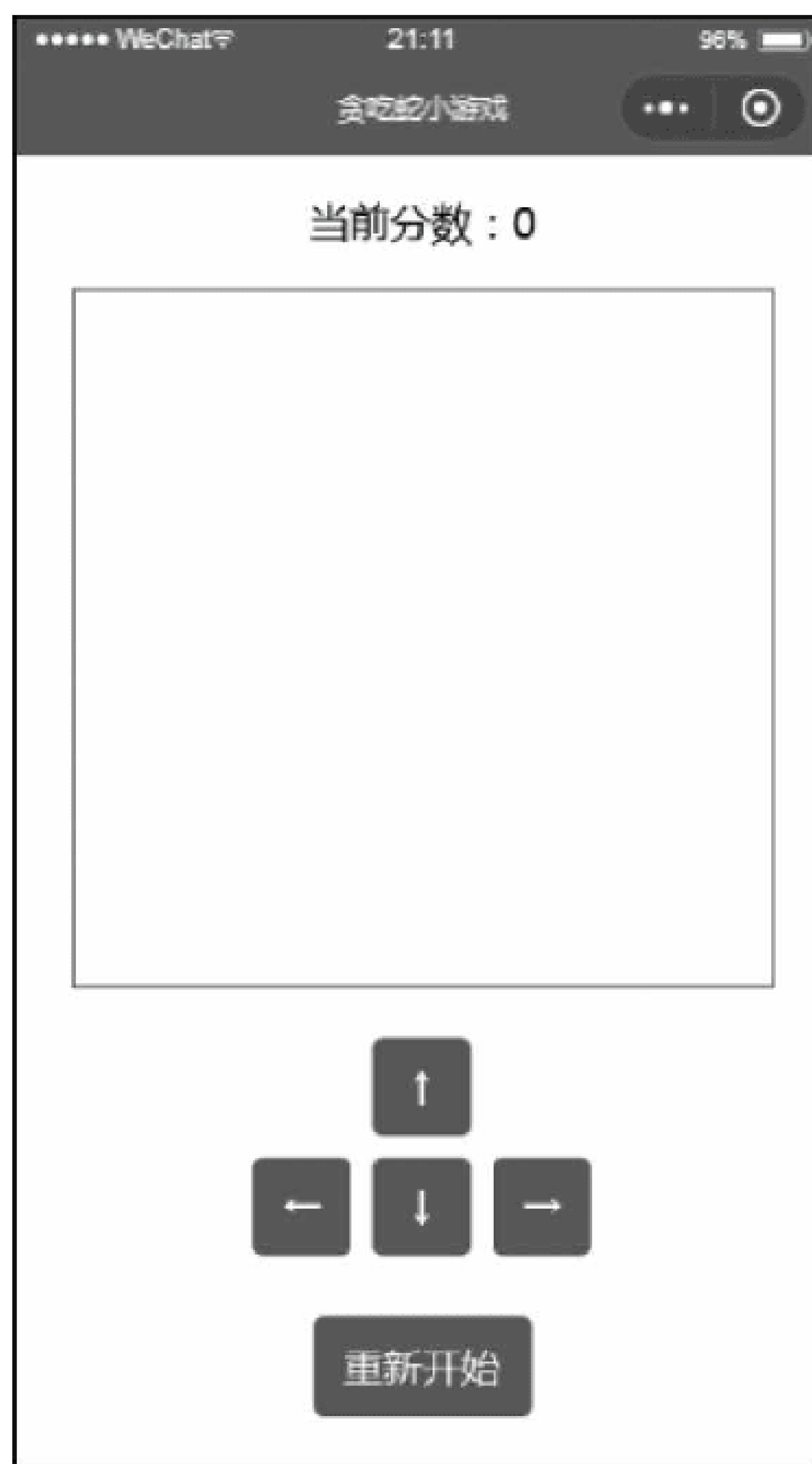


图 14-11　游戏页面效果图

由图可见，此时可以显示完整的布局样式。由于尚未绘制画布上的内容，所以暂时看不到贪吃蛇的动画效果。

此时页面布局与样式设计就已完成，14.5 节将介绍如何进行数据模型设计。



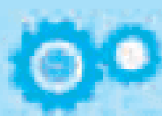
14.5　数据模型设计



本项目计划将游戏画布分割成 16 行、16 列的网格，每个网格的长、宽均为 20 像素。画布上的贪吃蛇是由一系列连续的网格填充颜色组成的，食物是由单个网格填充颜色而成的，因此只要知道了这些需要填色的网格坐标即可在画布上绘制出蛇身与食物。

14.5.1　贪吃蛇模型设计

本项目设置贪吃蛇的初始身长为 3 格，以贪吃蛇的蛇头出现在最左侧第二行并且向右移动为例，动态过程如图 14-12 所示。



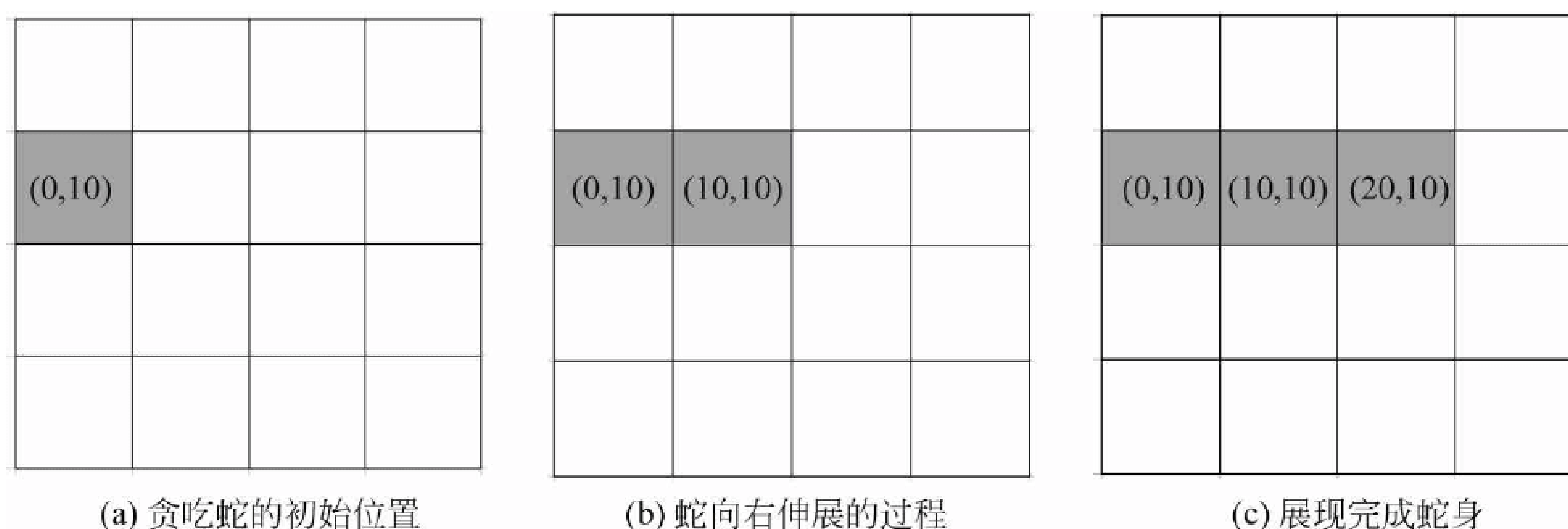


图 14-12 贪吃蛇的模型概念图

图中标记的所有坐标数据对应的都是网格的左上角坐标位置。由图 14-12(a)可见，本示例中贪吃蛇的初始位置出现在坐标(0,10)处，使用浅蓝色填充边长为 10 像素的网格来表示蛇身。由于目前规定为向右移动，所以随着每次游戏内容刷新都追加填充右侧一个空白网格作为蛇身，直至完整蛇身在网格中全部显现，这一过程由图 14-12(b)和图 14-12(c)所呈现。

使用一个数组 snakeMap 来记录组成蛇身的每一个网格坐标，并依次在画布的指定位置填充颜色，即可形成贪吃蛇从出现到移动直至展现完整身体的过程。

例如上面示例中向右移动的贪吃蛇初始状态的坐标可以记录为：

```
snakeMap = [{'x':0, 'y':10}];
```

随着蛇头向右前进，该数组增加第二组坐标：

```
snakeMap = [{'x':0, 'y':10}, {'x':10, 'y':10}];
```

如果继续向右前进，该数组增加第三组坐标：

```
snakeMap = [{'x':0, 'y':10}, {'x':10, 'y':10}, {'x':20, 'y':10}];
```

此时蛇身已经完整显示出来。

14.5.2 蛇身移动模型

当蛇身已经完全显示在游戏画面中时，如果蛇继续前进，则需要清除蛇尾的网格颜色，以表现出蛇的移动效果。以 14.5.1 节的贪吃蛇模型为例，分别展示其向右、向下和向上移动的效果，如图 14-13 所示。

图 14-13(b)显示的是贪吃蛇的蛇身已全部显示出来后仍继续向右前进的效果。在吃到食物之前，蛇的身长都将保持不变。此时除了需要填充右侧一个新的空白网格外还需要清除最早的一个蛇身网格颜色，以实现蛇在移动的动画效果。图 14-13(c)与图 14-13(d)显示的是贪吃蛇分别向下和向上移动的效果，其原理与图 14-13(b)的解释相同。

以 14.5.1 节介绍的数组 snakeMap 为例，继续讲解如何实现蛇的移动效果。

例如继续往右前进，该数组添加新坐标，并且还需要删除最早的一组坐标：

```
snakeMap = [{'x':10, 'y':10}, {'x':20, 'y':10}, {'x':30, 'y':10}];
```

因为该数组中的坐标只用于显示当前的蛇身数据，所以需要去掉曾经路过的轨迹。这种绘制方式可以展现贪吃蛇的动态移动效果。故只要每次游戏界面刷新时保持更新 snakeMap

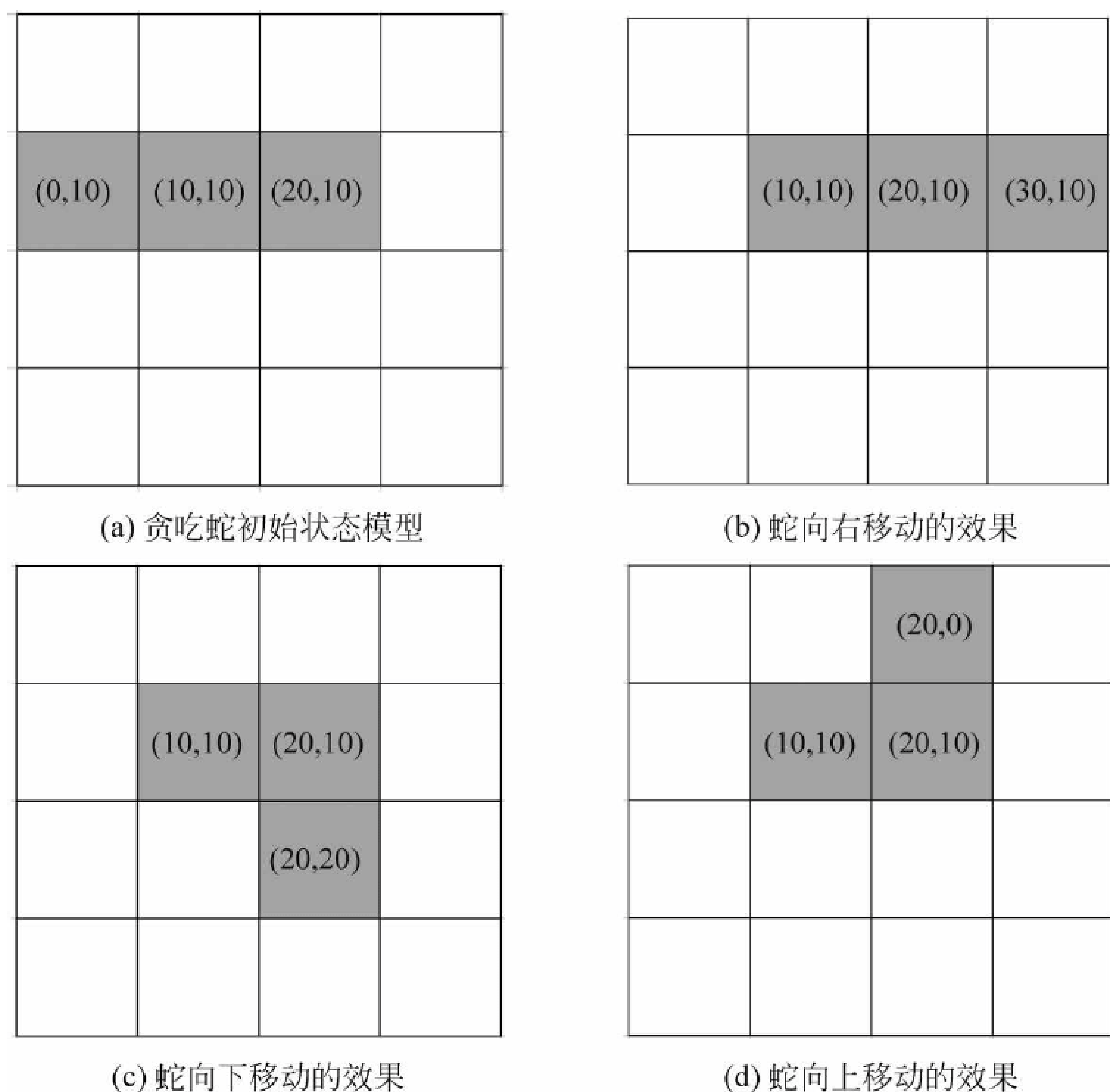


图 14-13　贪吃蛇的移动模型图

数组的记录即可获得贪吃蛇的当前位置。

14.5.3　蛇吃食物模型

在贪吃蛇的移动过程中，如果蛇头碰撞到食物，则认为蛇将食物吃掉了。此时蛇的身长增加一格，并且食物消失，然后在随机位置重新出现。同样以初始方格左上角位置在(0,10)坐标、身长为3格的蛇模型为例展示蛇吃食物的过程，如图14-14所示。

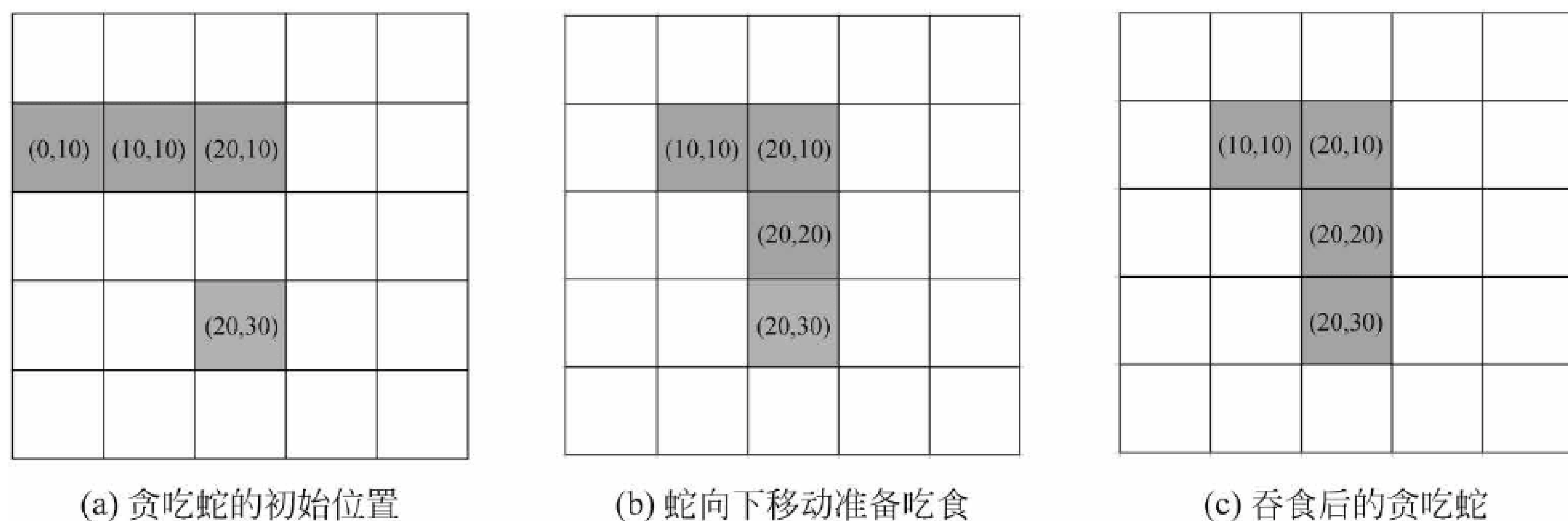
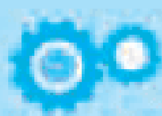


图 14-14　蛇吃食物的模型图

由图14-14(a)可见，食物位于蛇头的正下方，因此需要控制蛇头向下移动来接近食物；图14-14(b)显示的是蛇向下移动一格后的效果，此时蛇头已经贴在食物边上了；图14-14(c)显示的是食物被吞食后的效果，此时贪吃蛇的蛇身长度增加了一格并且食物消失。

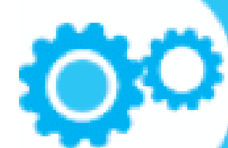
因此每当蛇吃到食物时，需要将表示蛇身的变量t自增1。然后判断用于记录蛇身坐标的



数组 snakeMap 的长度，如果与当前蛇身长度 t 的值相同，则不必删除最前面的数据。

此时的 snakeMap 数组坐标为：

```
snakeMap = [{'x':10, 'y':10}, {'x':20, 'y':10}, {'x':20, 'y':20}, {'x':20, 'y':30}];
```



14.6 逻辑实现



14.6.1 首页逻辑



视频讲解

首页主要需要实现点击图片能跳转到游戏页面，并且根据游戏难度模式规定贪吃蛇的行动速度。本项目根据贪吃蛇的行动速度快慢计划采用两种游戏模式，即简单模式(每 0.5s 行动一次)和困难模式(每 0.2s 行动一次)。

相关 WXML(pages/index/index.wxml)代码片段修改如下：

```
1.  <view class="container">
2.    <image src='/images/snake01.png' bindtap='goToGame' data-level='easy'></image>
3.    <image src='/images/snake02.png' bindtap='goToGame' data-level='hard'></image>
4.  </view>
```

上述代码表示为关卡添加了自定义点击事件函数 goToGame，并且使用 data-level 属性携带了游戏难度模式信息 easy 和 hard。

然后在对应的 index.js 文件中添加 goToGame 函数的内容，代码片段如下：

```
1.  Page({
2.    /**
3.     * 自定义函数--跳转游戏页面
4.     */
5.    goToGame: function(e) {
6.      //获取游戏模式
7.      let level = e.currentTarget.dataset.level
8.      //游戏界面刷新的间隔时间，单位为毫秒(数字越大，蛇的速度越慢)
9.      let time = 0
10.
11.     //简单模式
12.     if (level == 'easy') {
13.       time = 500
14.     }
15.     //困难模式
16.     else if (level == 'hard') {
17.       time = 200
18.     }
19.
20.     //跳转到游戏页面
21.     wx.navigateTo({
22.       url: '../game/game?time=' + time,
23.     })
24.   },
25. })
```

此时已经可以点击跳转到 game 页面，并且成功携带了贪吃蛇的行动速度数据，但是仍需在 game 页面进行携带数据的接收处理才可显示正确的游戏画面。

14.6.2 游戏页逻辑

游戏页面主要有以下几个功能需要实现：

- 初始化游戏数据，包括蛇身长度、游戏速度、初始食物位置等；
- 游戏画面的绘制；
- 4个方向键可以改变贪吃蛇的行动方向；
- 点击"重新开始"按钮可以使游戏回到初始状态；
- 游戏分数的记录；
- 游戏失败的判断。

1 准备工作



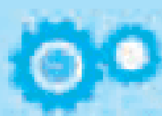
视频讲解

首先在 game.js 顶端使用一系列数据表示贪吃蛇的初始状态，包括蛇身长度、首次出现的位置和移动方向等。相关 JS(pages/game/game.js)代码片段如下：

```
1.  //=============
2.  //游戏参数设置
3.  //=============
4.  //蛇的身长
5.  var t = 3
6.  //记录蛇的运动轨迹，用数组记录每个坐标点
7.  var snakeMap = []
8.  //蛇身单元格大小
9.  var w = 20
10. //方向代码：上为1、下为2、左为3、右为4
11. var direction = 1
12. //蛇的初始坐标
13. var x = 0
14. var y = 0
15. //食物的初始坐标
16. var foodX = 0
17. var foodY = 0
18. //画布的宽和高
19. var width = 320
20. var height = 320
21. //游戏界面刷新的间隔时间，单位为毫秒(数字越大，蛇的速度越慢)
22. var time = 1000
23.
24. Page({
25.   /**
26.    * 页面的初始数据
27.    */
28.   data: {
29.     score: 0                    //游戏当前分数
30.   },
31. })
```

需要注意的是，此时 time 只是初始值，还需要在 game.js 的 onLoad 函数中读取从首页传递来的参数值对其进一步更新。相关 JS(pages/game/game.js)代码片段如下：

```
1.  Page({
2.    /**
```



```
3.      * 生命周期函数 -- 监听页面加载
4.      */
5.    onLoad: function(options) {
6.      //更新游戏刷新时间
7.      time = options.time
8.    },
9.  })
```

另外还需要在 game.js 的 onLoad 函数中对画布上下文进行初始化，以便后续可以进行贪吃蛇和食物的绘制工作。相关 JS(pages/game/game.js)代码片段如下：

```
1.  Page({
2.    /**
3.      * 生命周期函数 -- 监听页面加载
4.      */
5.    onLoad: function(options) {
6.      //更新游戏刷新时间
7.      time = options.time
8.
9.      //创建画布上下文
10.     this.ctx = wx.createCanvasContext('myCanvas')
11.   },
12. })
```

最后修改 game.wxml 页面，将游戏分数用{{score}}表示，相关 WXML(pages/game/game.wxml)代码如下：

```
1.  <view class = 'container'>
2.    <!-- 关卡提示 -->
3.    <view>当前分数: {{score}} </view>
4.
5.    <!-- 游戏画布(代码略) -->
6.    <!-- 方向键(代码略) -->
7.    <!-- 重新开始按钮(代码略) -->
8.  </view>
```

2 绘制贪吃蛇逻辑实现

1) 初始化贪吃蛇数据



视频讲解

每次游戏重新开始时需要重新初始化贪吃蛇的一系列数据，例如蛇身长度、蛇身坐标、蛇头坐标和前进方向等。

在 game.js 文件中创建自定义函数 gameStart，用于启动游戏，对应的 JS(pages/game/game.js)代码片段添加如下：

```
1.  Page({
2.    /**
3.     * 自定义函数 -- 启动游戏
4.     */
5.    gameStart: function() {
6.      //初始化蛇身长度
7.      t = 3
8.      //初始化蛇身坐标
9.      snakeMap = []
10.
11.     //随机生成贪吃蛇的初始蛇头坐标
```

```
12.       x = Math.floor(Math.random() * width / w) * w
13.       y = Math.floor(Math.random() * height / w) * w
14.
15.       //随机生成蛇的初始前进方向
16.       direction = Math.ceil(Math.random() * 4)
17.     },
18. })
```

若蛇的初始位置与方向的初始数据均为固定值，则会降低游戏的难度和可玩度，因此可以使用随机数重新定义贪吃蛇初始出现的位置和移动方向，以便每次游戏都可以获得不同的效果。

2）绘制蛇身


视频讲解

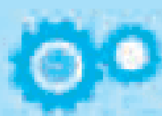
每次游戏画面刷新，蛇都需要在指定方向上再前进一格。如果蛇没有吃到新的食物，则还需要清除原先蛇尾最后一个位置的颜色，以表现出贪吃蛇动态前进了一格的效果。

在 game.js 中声明自定义函数 drawSnake，专门用于绘制贪吃蛇的蛇身，对应的 JS(pages/game/game.js)代码片段如下：

```
1.  Page({
2.    /**
3.     * 自定义函数 -- 绘制贪吃蛇
4.     */
5.    drawSnake: function() {
6.      let ctx = this.ctx
7.      //设置蛇身的填充颜色
8.      ctx.setFillStyle('lightblue')
9.      //绘制全部蛇身
10.     for (var i = 0; i < snakeMap.length; i++) {
11.       ctx.fillRect(snakeMap[i].x, snakeMap[i].y, w, w)
12.     }
13.   },
14. })
```

由于要通过游戏画面刷新才能实现动画效果，所以在 game.js 中声明自定义函数 gameRefresh 专门用于刷新画布，并在该函数中调用 drawSnake 函数来绘制贪吃蛇的蛇身变化过程。对应的 JS(pages/game/game.js)代码片段如下：

```
1.  Page({
2.    /**
3.     * 自定义函数 -- 游戏画面刷新
4.     */
5.    gameRefresh: function() {
6.      //将当前坐标添加到贪吃蛇的运动轨迹坐标数组中
7.      snakeMap.push({
8.        'x': x,
9.        'y': y
10.     })
11.
12.     //数组只保留蛇身长度的数据，如果蛇前进了，则删除最旧的坐标
13.     if (snakeMap.length > t) {
14.       snakeMap.shift()
15.     }
16.
17.     //绘制贪吃蛇
```



```
18.        this.drawSnake()
19.
20.        //在画布上绘制全部内容
21.        this.ctx.draw()
22.
23.        //根据方向移动蛇头的下一个位置
24.        switch (direction) {
25.          case 1:
26.            y -= w
27.            break
28.          case 2:
29.            y += w
30.            break
31.          case 3:
32.            x -= w
33.            break
34.          case 4:
35.            x += w
36.            break
37.        }
38.      },
39.  })
```

然后在 game.js 中修改自定义函数 gameStart，使用 setInterval()方法设置在间隔规定的时间后重复调用 gameRefresh 已达到游戏画面刷新的效果。修改后的 gameStart 函数如下：

```
1.   Page({
2.     /**
3.      * 自定义函数 -- 启动游戏
4.      */
5.     gameStart: function() {
6.       //初始化蛇身长度(代码略)
7.       //初始化蛇身坐标(代码略)
8.       //随机生成贪吃蛇的初始蛇头坐标(代码略)
9.       //随机生成蛇的初始前进方向(代码略)
10.
11.      //每隔 time 毫秒刷新一次游戏内容
12.      var that = this
13.      this.interval = setInterval(function() {
14.        that.gameRefresh()
15.      }, time)
16.    },
17.  })
```

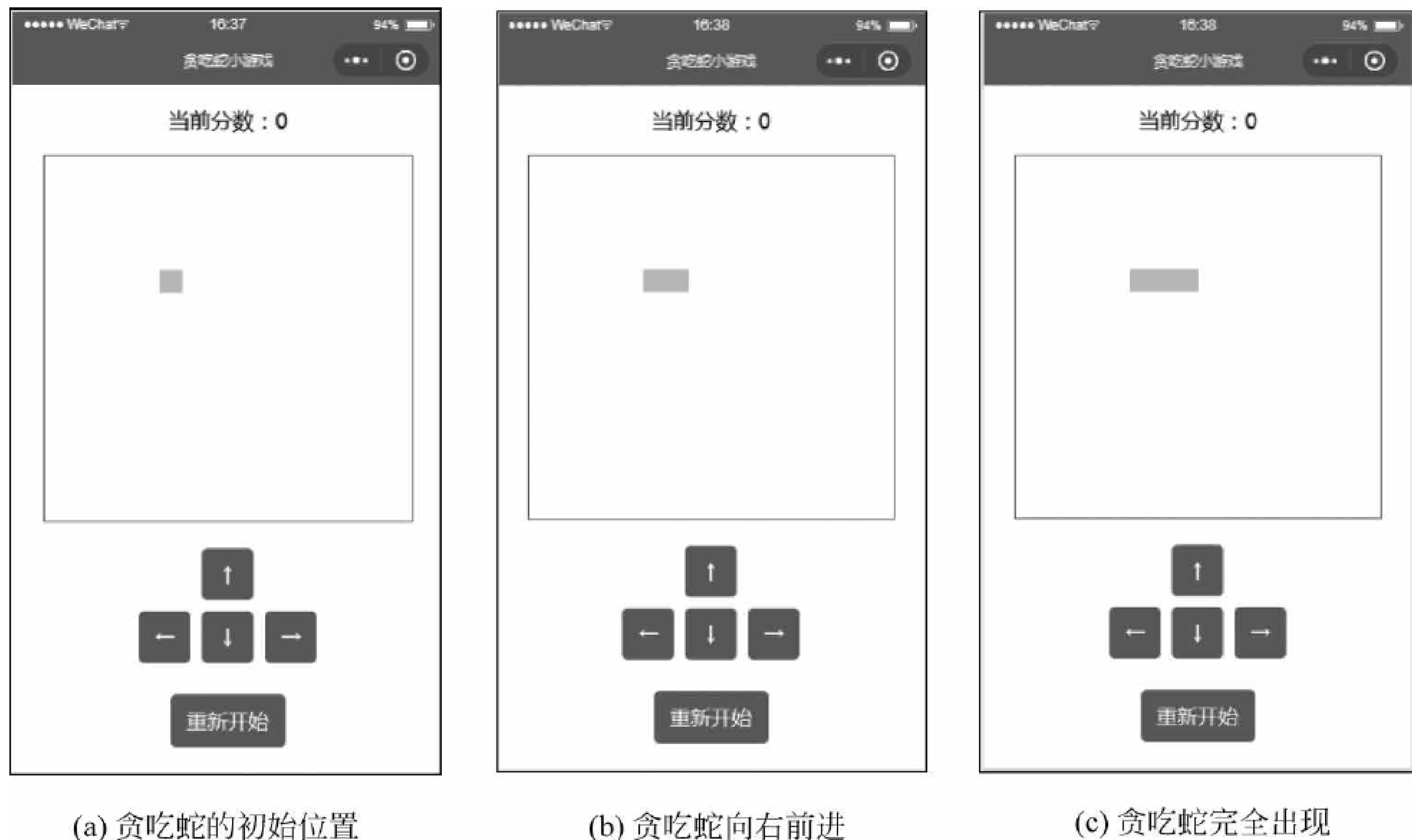
最后在 game.js 的 onLoad 函数中调用 gameStart，使动画效果启动。

修改后的 onLoad 函数如下：

```
1.   Page({
2.     /**
3.      * 生命周期函数 -- 监听页面加载
4.      */
5.     onLoad: function(options) {
6.       //更新游戏刷新时间(代码略)
7.       //创建画布上下文(代码略)
8.
```

```
9.        //开始游戏
10.       //this.gameStart()
11.    },
12. })
```

运行效果如图 14-15 所示。



(a) 贪吃蛇的初始位置 (b) 贪吃蛇向右前进 (c) 贪吃蛇完全出现

图 14-15 贪吃蛇的蛇身绘制效果

3）方向键逻辑实现



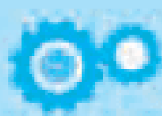
视频讲解

贪吃蛇是依靠玩家按画面上的方向键进行上、下、左、右方向切换的，因此修改 game.wxml 页面中的 4 个方向键<button>，为其绑定点击事件。

WXML(pages/game/game.wxml)代码修改后如下：

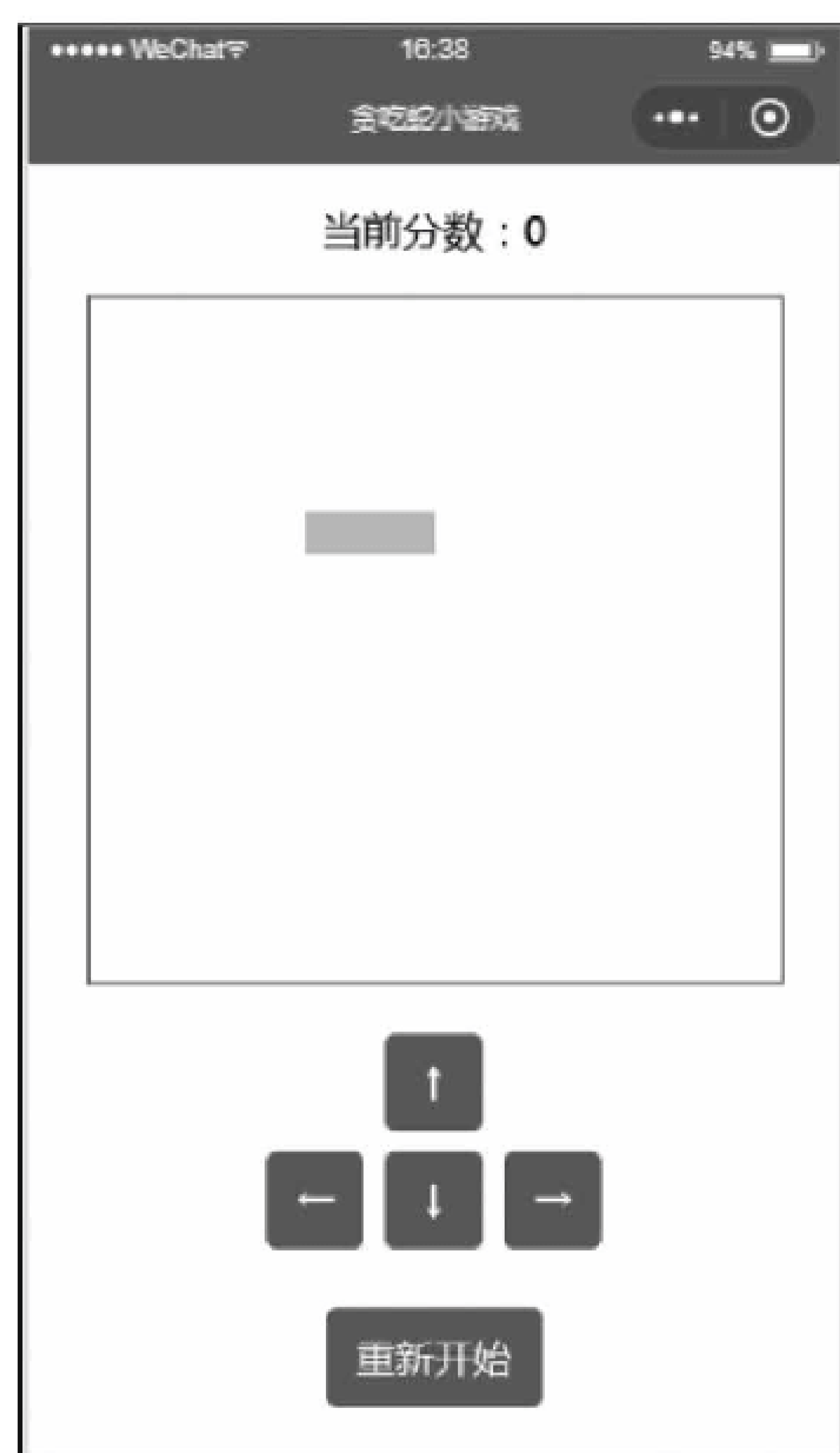
```
1.  <view class = 'container'>
2.    <!-- 当前分数提示(代码略) -->
3.
4.    <!-- 游戏画布(代码略) -->
5.
6.    <!-- 方向键 -->
7.    <view class = 'btnBox'>
8.      <button bindtap = 'up'>↑</button>
9.      <view>
10.       <button bindtap = 'left'>←</button>
11.       <button bindtap = 'down'>↓</button>
12.       <button bindtap = 'right'>→</button>
13.     </view>
14.   </view>
15.
16.   <!--"重新开始"按钮(代码略) -->
17. </view>
```

在 game.js 文件中添加自定义函数 up、down、left 和 right，分别用于实现贪吃蛇向上、下、左、右 4 个方向的移动。对应的 JS(pages/game/game.js)代码片段如下：

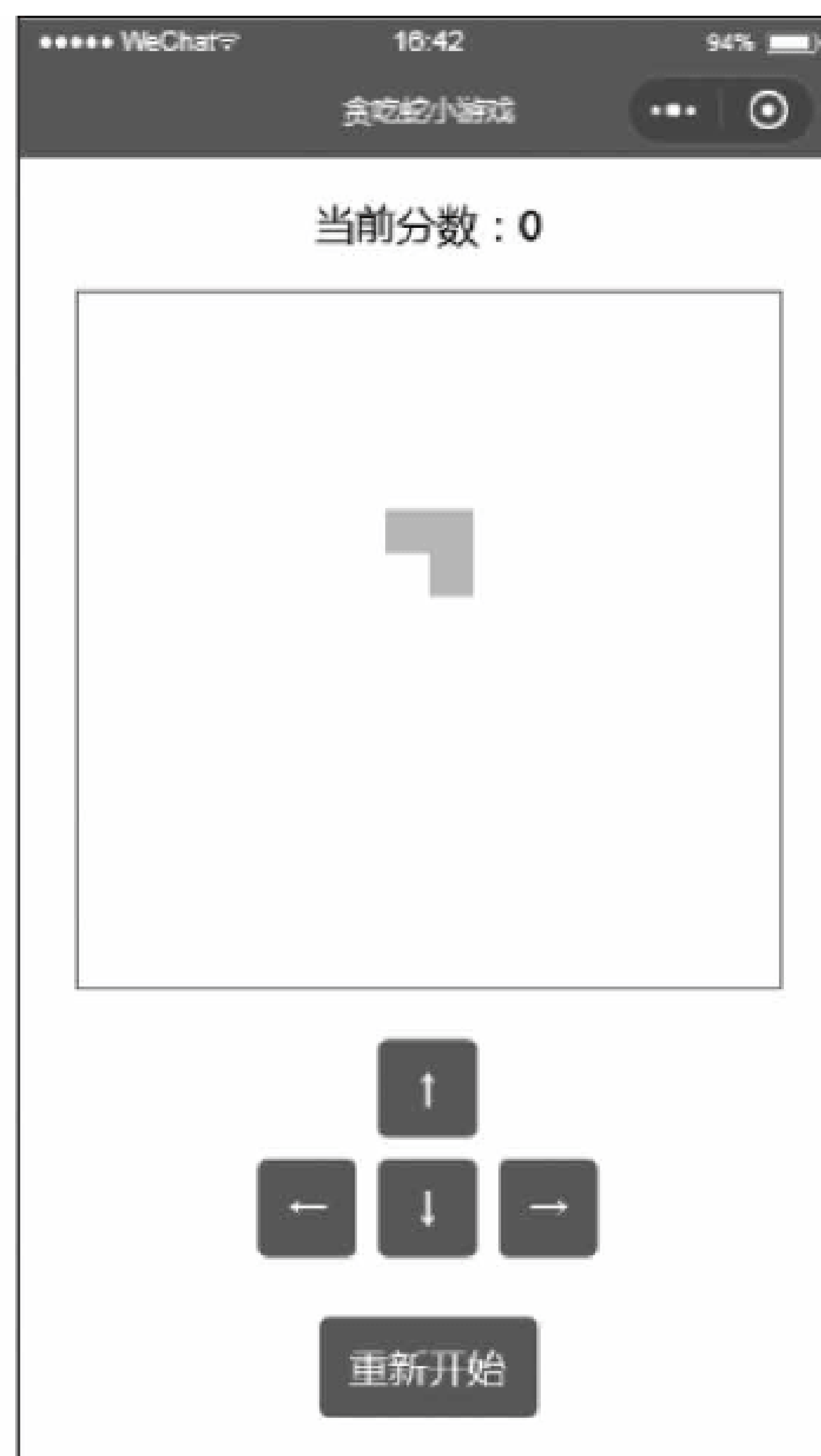


```
1.  Page({
2.    /**
3.     * 自定义函数--监听方向键：上
4.     */
5.    up: function() {
6.      direction = 1
7.    },
8.
9.    /**
10.    * 自定义函数--监听方向键：下
11.    */
12.   down: function() {
13.     direction = 2
14.   },
15.
16.   /**
17.    * 自定义函数--监听方向键：左
18.    */
19.   left: function() {
20.     direction = 3
21.   },
22.
23.   /**
24.    * 自定义函数--监听方向键：右
25.    */
26.   right: function() {
27.     direction = 4
28.   },
29. })
```

运行效果如图 14-16 所示。



(a) 贪吃蛇向右前进



(b) 贪吃蛇改变方向

图 14-16 贪吃蛇的方向改变

3 绘制食物逻辑实现

接下来需要在画布上为贪吃蛇绘制食物。食物每次将在随机网格位置出现，占一格位置。食物每次只在画面中呈现一个，直到被蛇头触碰表示吃掉方可在原先的位置清除，并在下一个随机位置重新产生。

1）初始化食物数据



视频讲解

对应的 JS(pages/game/game.js)代码片段添加如下：

```
1.  Page({
2.  /**
3.     * 自定义函数 -- 启动游戏
4.     */
5.    gameStart: function() {
6.      //初始化蛇身长度(代码略)
7.      //初始化蛇身坐标(代码略)
8.      //随机生成贪吃蛇的初始蛇头坐标(代码略)
9.      //随机生成蛇的初始前进方向(代码略)
10.
11.     //随机生成食物的初始坐标
12.     //foodX = Math.floor(Math.random() * width / w) * w
13.     //foodY = Math.floor(Math.random() * height / w) * w
14.
15.     //每隔 time 毫秒刷新一次游戏内容(代码略)
16.   },
17. })
```

2）绘制食物



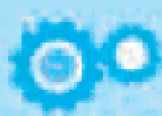
视频讲解

在 game.js 中声明自定义函数 drawFood，专门用于绘制食物，对应的 JS(pages/game/game.js)代码片段如下：

```
1.  Page({
2.    /**
3.     * 自定义函数 -- 绘制食物
4.     */
5.    drawFood: function() {
6.      let ctx = this.ctx
7.      //设置食物的填充颜色
8.      ctx.setFillStyle('red')
9.      //绘制食物
10.     ctx.fillRect(foodX, foodY, w, w)
11.   },
12. })
```

然后在 game.js 中修改自定义函数 gameRefresh，在该函数中调用 drawFood 函数在指定的位置绘制食物。对应的 JS(pages/game/game.js)代码片段如下：

```
1.  Page({
2.    /**
3.     * 自定义函数 -- 游戏画面刷新
4.     */
5.    gameRefresh: function() {
6.      //在随机位置绘制一个食物
7.      this.drawFood()
8.
```



```
9.        //将当前坐标添加到贪吃蛇的运动轨迹坐标数组中(代码略)
10.       //数组只保留蛇身长度的数据,如果蛇前进了,则删除最旧的坐标(代码略)
11.       //绘制贪吃蛇(代码略)
12.       //在画布上绘制全部内容(代码略)
13.       //根据方向移动蛇头的下一个位置(代码略)
14.     },
15. })
```

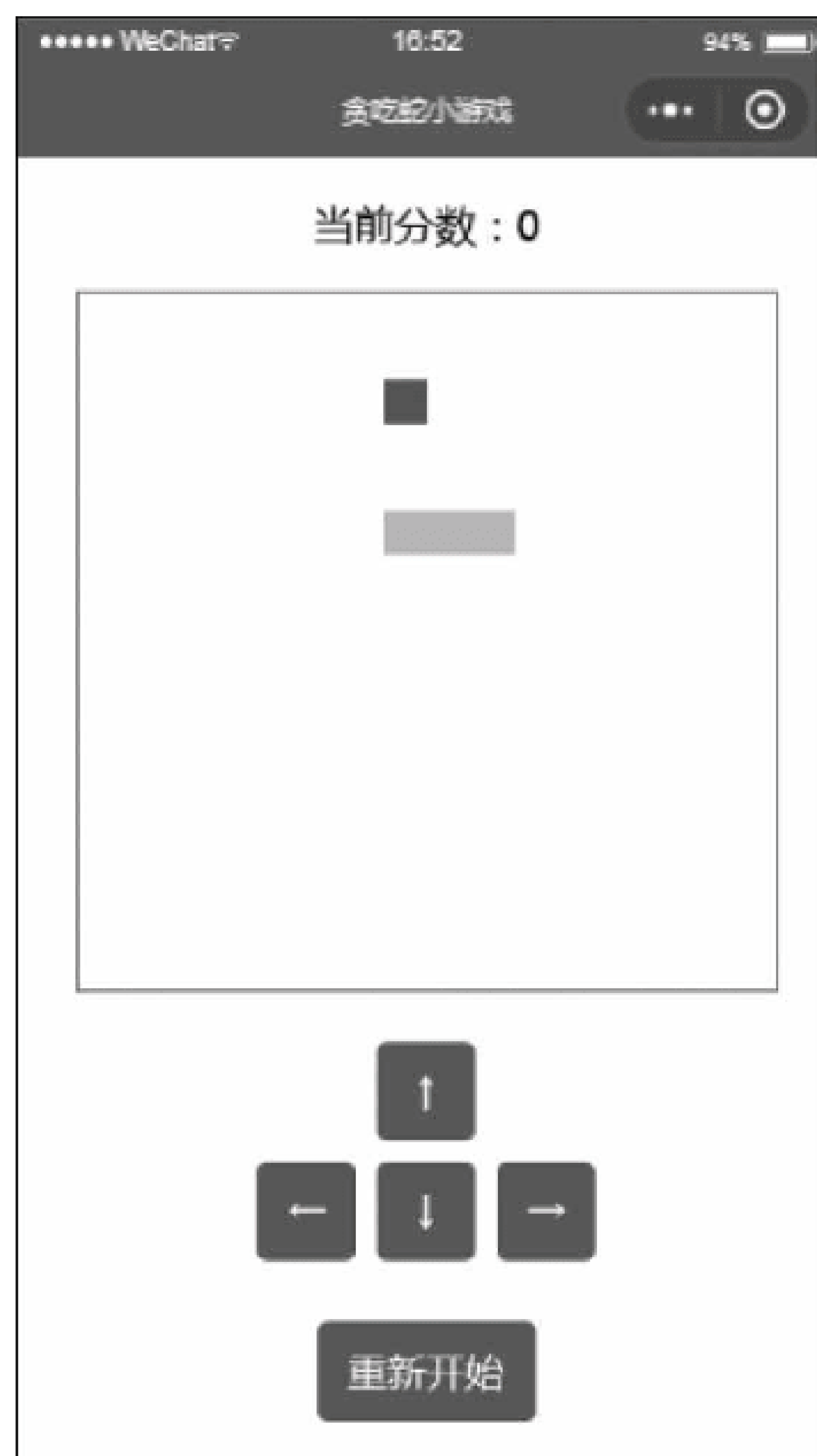


图 14-17　贪吃蛇和食物同时出现

运行效果如图 14-17 所示。

3）吃到食物的判定

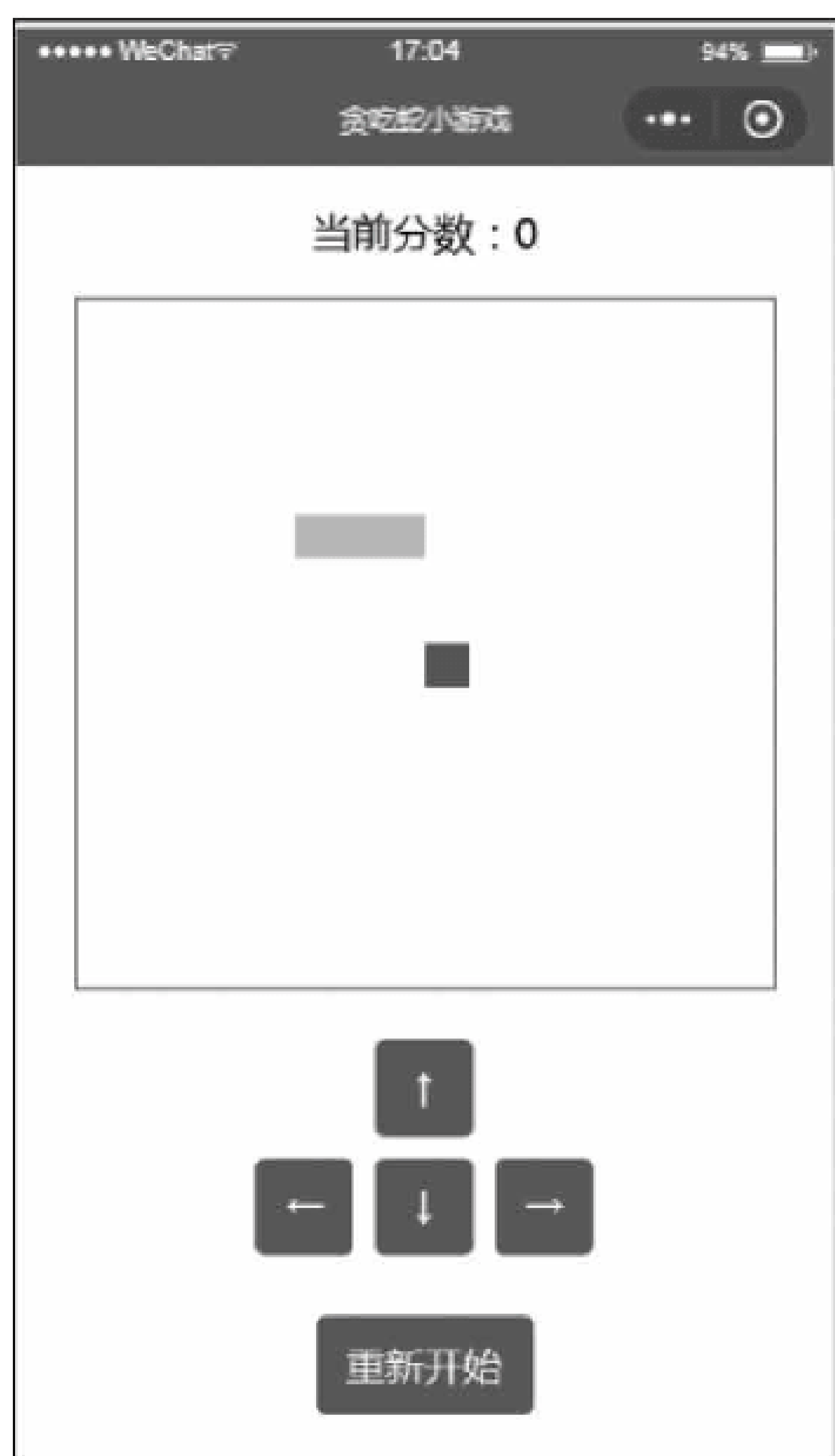

视频讲解

当蛇头和食物出现在同一个方格中时判定蛇吃到了食物，此时食物消失、当前分数增加 10 分、蛇身增加一格，并且在随机位置重新生成下一个食物。

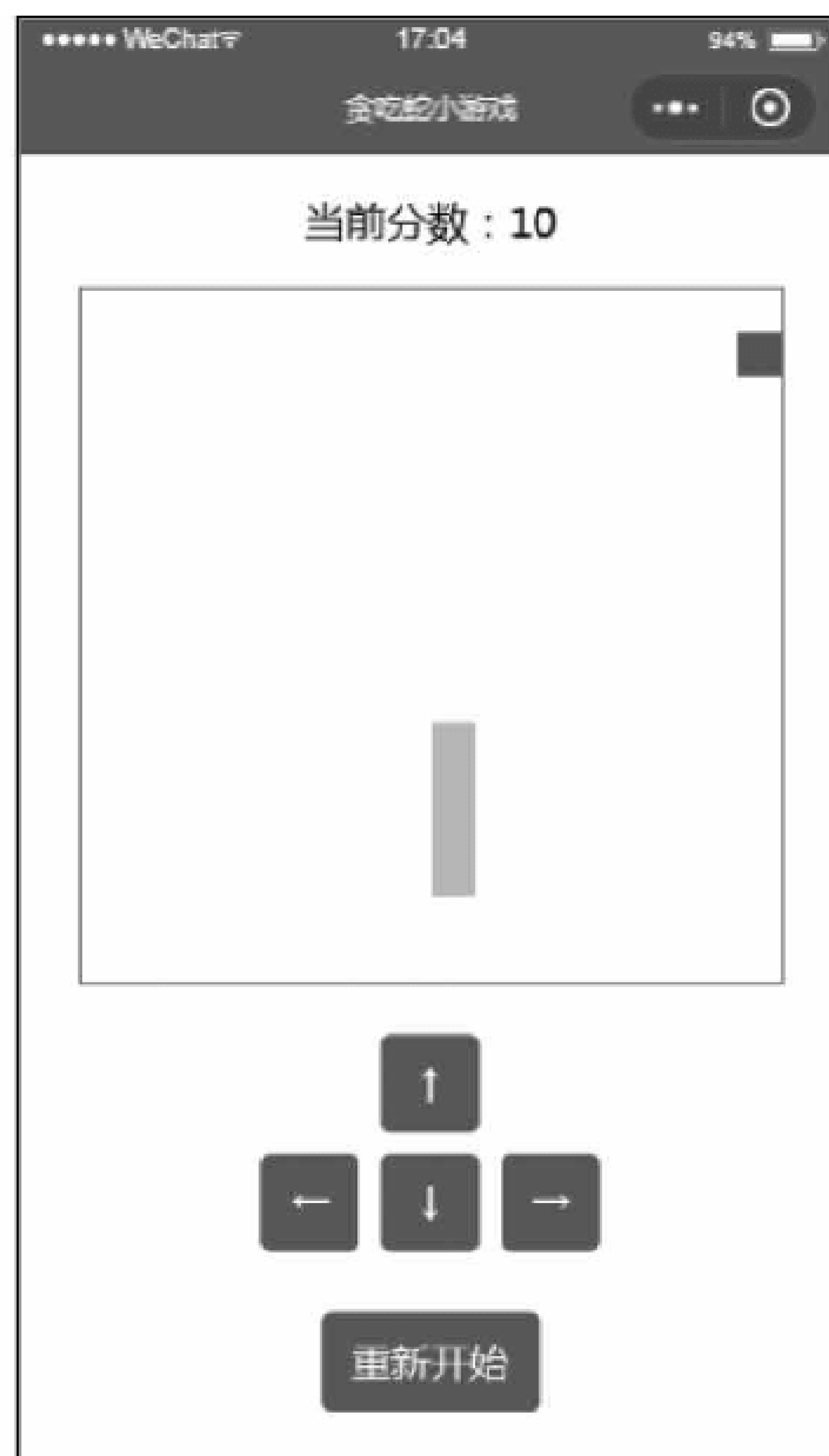
修改自定义函数 gameRefresh，对应的 JS(pages/game/game.js)代码片段添加如下：

```
1.  Page({
2.  /**
3.     * 自定义函数 -- 游戏画面刷新
4.     */
5.    gameRefresh: function() {
6.      //在随机位置绘制一个食物(代码略)
7.      //将当前坐标添加到贪吃蛇的运动轨迹坐标数组中
    (代码略)
8.      //数组只保留蛇身长度的数据,如果蛇前进了,则删除最旧的坐标(代码略)
9.      //绘制贪吃蛇(代码略)
10.
11.     //吃到食物的判定
12.     if (foodX == x && foodY == y) {
13.       //吃到一次食物加 10 分
14.       let score = this.data.score + 10
15.       this.setData({
16.         score: score
17.       })
18.
19.       //随机生成下一个食物的初始坐标
20.       foodX = Math.floor(Math.random() * width / w) * w
21.       foodY = Math.floor(Math.random() * height / w) * w
22.
23.       //在新的随机位置绘制食物
24.       this.drawFood()
25.
26.       //蛇身长度加 1
27.       t++
28.     }
29.
30.     //在画布上绘制全部内容(代码略)
31.     //根据方向移动蛇头的下一个位置(代码略)
32.   },
33. })
```

运行效果如图 14-18 所示。



(a) 贪吃蛇向右前进



(b) 贪吃蛇吃到第一个食物

图 14-18　贪吃蛇的方向改变

其中图 14-18(a)是尚未吃过食物的贪吃蛇向右前进的画面，此时蛇身为 3 个网格长度、当前分数为 0 分；图 14-18(b)是贪吃蛇已经吃到了第一个食物的画面，此时蛇身长度增加 1 格、当前分数更新为 10 分，并且在新的位置出现了食物。

4 碰撞检测逻辑实现



视频讲解

如果蛇头撞到了游戏画面的任意一边或者撞到蛇身均判定为游戏失败，此时弹出提示对话框告知玩家游戏失败的原因，并提示其重新开始。玩家点击对话框上的“确定”按钮则可以开始下一局游戏。

在 game.js 中创建自定义函数 detectCollision 用于进行蛇与障碍物的碰撞检测。碰撞检测需要检测两种可能性，一是蛇头撞到了四周的墙壁，二是蛇头撞到了蛇身。无论哪一种情况发生都判定游戏失败。

对应的 JS(pages/game/game.js)代码片段添加如下：

```
1.  Page({
2.    /**
3.     * 自定义函数 -- 碰撞检测
4.     */
5.    detectCollision: function() {
6.      //如果蛇头撞到了四周的墙壁,游戏失败
7.      if (x > width || y > height || x < 0 || y < 0) {
8.        return 1
9.      }
10.
11.     //如果蛇头撞到了蛇身,游戏失败
12.     for (var i = 0; i < snakeMap.length; i++) {
13.       if (snakeMap[i].x == x && snakeMap[i].y == y) {
14.         return 2
15.       }
```

```
16.       }
17.
18.       //没有碰撞
19.       return 0
20.     },
21.  })
```

该函数具有 3 种返回值,分别表示不同的含义。

- 返回值为 0:表示本次没有碰撞到障碍物,游戏继续;
- 返回值为 1:表示蛇头碰撞到了游戏画布任意一边的墙壁,游戏失败;
- 返回值为 2:表示蛇头碰撞到了蛇身,游戏失败。

然后在 game.js 中修改 gameRefresh 函数,要求一旦游戏失败则弹出提示对话框。

对应的 JS(pages/game/game.js)代码片段修改如下:

```
1.   Page({
2.   /**
3.     * 自定义函数 -- 游戏画面刷新
4.     */
5.     gameRefresh: function() {
6.       //在随机位置绘制一个食物(代码略)
7.       //将当前坐标添加到贪吃蛇的运动轨迹坐标数组中(代码略)
8.       //数组只保留蛇身长度的数据,如果蛇前进了,则删除最旧的坐标(代码略)
9.       //绘制贪吃蛇(代码略)
10.      //吃到食物的判定(代码略)
11.      //在画布上绘制全部内容(代码略)
12.      //根据方向移动蛇头的下一个位置(代码略)
13.
14.      //碰撞检测,返回值为 0 表示没有撞到障碍物
15.      let code = this.detectCollision()
16.      if (code != 0) {
17.        //游戏停止
18.        clearInterval(this.interval)
19.
20.        var msg = ''
21.        if (code == 1) {
22.          msg = '失败原因:撞到了墙壁'
23.        } else if (code == 2) {
24.          msg = '失败原因:撞到了蛇身'
25.        }
26.
27.        wx.showModal({
28.          title: '游戏失败,是否重来?',
29.          content: msg,
30.          success: res => {
31.            if (res.confirm) {
32.              //重新开始游戏
33.              this.gameStart()
34.            }
35.          }
36.        })
37.      }
38.    },
39.  })
```

加上碰撞检测后的游戏运行效果如图 14-19 所示。



(a) 贪吃蛇碰撞到墙壁



(b) 贪吃蛇碰撞到蛇身

图 14-19　碰撞检测提示画面

5 重新开始游戏



视频讲解

修改 game.wxml 代码，为“重新开始”按钮追加自定义函数的点击事件。
WXML(pages/game/game.wxml)代码片段修改如下：

```
1.  <view class = 'container'>
2.    …
3.
4.    <!-- “重新开始”按钮 -->
5.    <button type = 'warn' bindtap = 'restartGame'>重新开始</button>
6.  </view>
```

在 game.js 文件中添加 restartGame 函数，用于重新开始游戏。
对应的 JS(pages/game/game.js)代码片段添加如下：

```
1.  Page({
2.    /**
3.     * 自定义函数 -- 重新开始游戏
4.     */
5.    restartGame: function() {
6.    clear Interval(this.interval)
7.      this.gameStart()
8.    },
9.  })
```

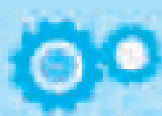
运行效果如图 14-20 所示。

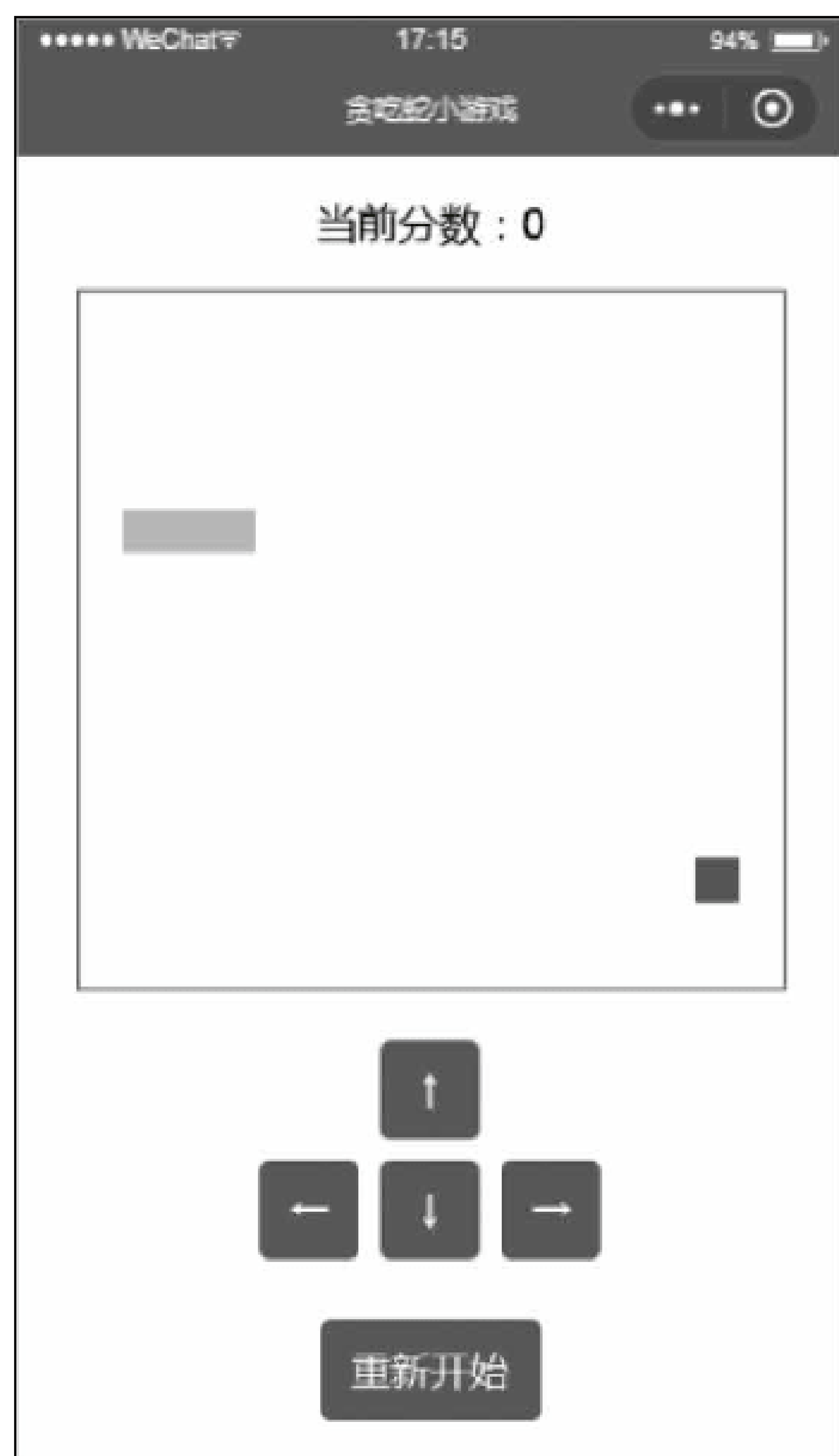
6 返回首页时停止游戏



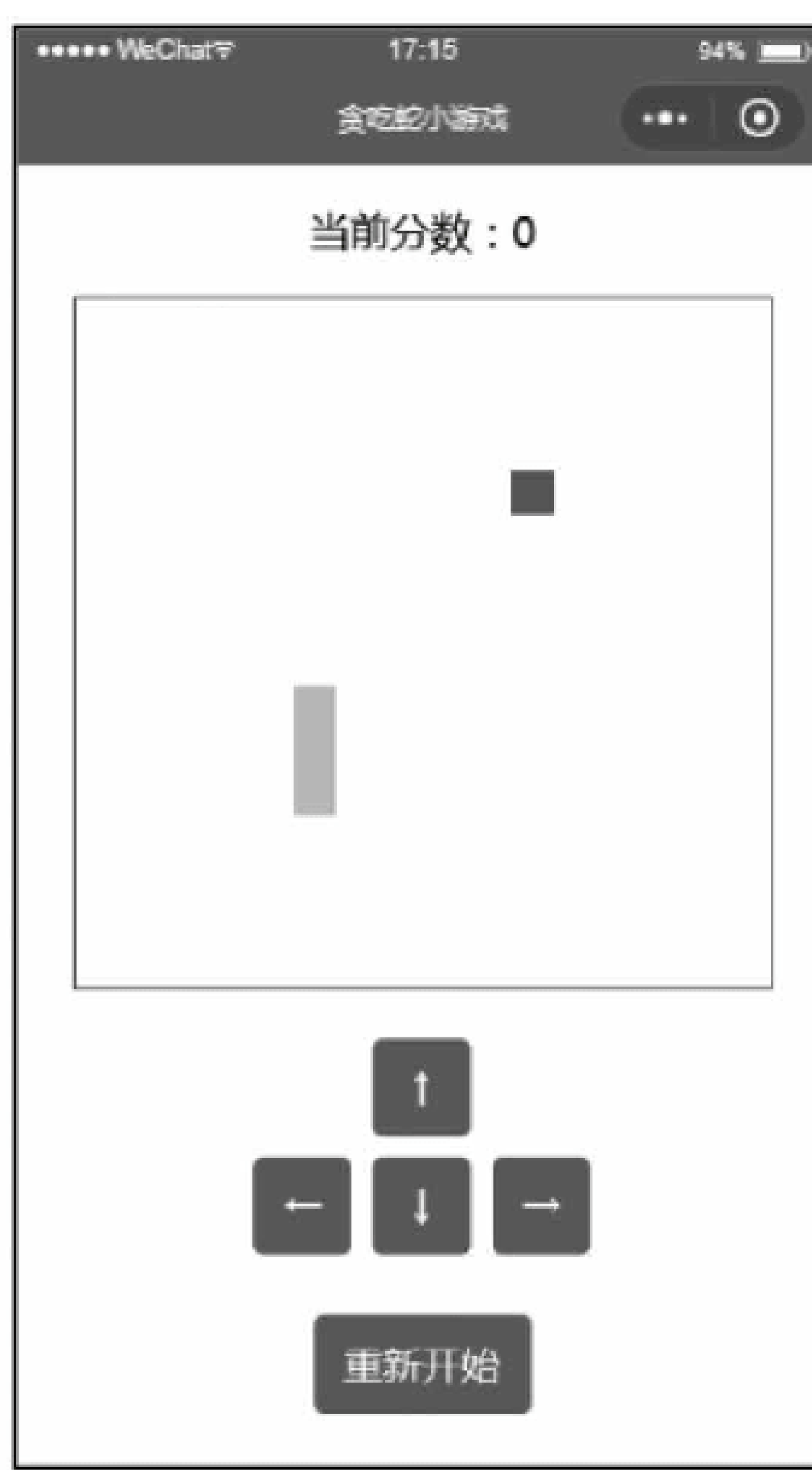
视频讲解

当游戏中途点击左上角返回键返回首页时还需要停止游戏，在 game.js 的 onUnload 函数中停止定时器即可。对应的 JS(pages/game/game.js)代码片段添加如下：



(a) 游戏初始运行画面

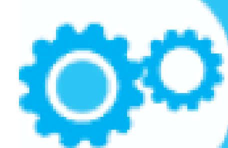


(b) 重新开始游戏画面

图 14-20 点击“重新开始”按钮的效果

```
1.  Page({
2.    /**
3.     * 生命周期函数--监听页面卸载
4.     */
5.    onUnload: function() {
6.      //游戏停止
7.      clearInterval(this.interval)
8.    },
9.  })
```

此时所有的代码内容就全部完成了。



14.7 完整代码展示



14.7.1 应用文件代码展示

1 app.json 代码展示

app.json 文件的完整代码如下：

```
1.  {
2.    "pages":[
3.      "pages/index/index",
4.      "pages/game/game"
5.    ],
6.    "window":{
7.      "navigationBarBackgroundColor": "#E64340",
8.      "navigationBarTitleText": "贪吃蛇小游戏"
9.    }
10. }
```

2 app.wxss 代码展示

app.wxss 文件的完整代码如下：

```
1.  /* 页面容器样式 */
2.  .container {
3.    height: 100vh;
4.    display: flex;
5.    flex-direction: column;
6.    align-items: center;
7.    justify-content: space-evenly;
8.  }
```

14.7.2 页面文件代码展示

1 首页代码展示

WXML 文件(pages/index/index.wxml)的完整代码如下：

```
1.  <view class="container">
2.    <image src='/images/snake01.png' bindtap='goToGame' data-level='easy'></image>
3.    <image src='/images/snake02.png' bindtap='goToGame' data-level='hard'></image>
4.  </view>
```

WXSS 文件(pages/index/index.wxss)的完整代码如下：

```
1.  /* 关卡图片 */
2.  image{
3.    width: 400rpx;
4.    height: 400rpx;
5.  }
```

JS 文件(pages/index/index.js)的完整代码如下：

```
1.  Page({
2.    /**
3.     * 自定义函数--跳转游戏页面
4.     */
5.    goToGame: function(e) {
6.      //获取游戏模式
7.      let level = e.currentTarget.dataset.level
8.      //游戏界面刷新的间隔时间,单位为毫秒(数字越大,蛇的速度越慢)
9.      let time = 0
10.
11.     //简单模式
12.     if (level == 'easy') {
13.       time = 500
14.     }
15.     //困难模式
16.     else if (level == 'hard') {
17.       time = 200
18.     }
19.
20.     //跳转到游戏页面
21.     wx.navigateTo({
22.       url: '../game/game?time=' + time,
```

```
23.       })
24.     },
25.  })
```

2 游戏页代码展示

WXML 文件(pages/game/game.wxml)的完整代码如下：

```
1.  <view class = 'container'>
2.    <!-- 关卡提示 -->
3.    <view>当前分数：{{score}}</view>
4.
5.    <!-- 游戏画布 -->
6.    <canvas canvas - id = 'myCanvas'></canvas>
7.
8.    <!-- 方向键 -->
9.    <view class = 'btnBox'>
10.     <button bindtap = 'up'>↑</button>
11.     <view>
12.       <button bindtap = 'left'>←</button>
13.       <button bindtap = 'down'>↓</button>
14.       <button bindtap = 'right'>→</button>
15.     </view>
16.   </view>
17.
18.   <!-- "重新开始"按钮 -->
19.   <button bindtap = 'restartGame'>重新开始</button>
20. </view>
```

WXSS 文件(pages/game/game.wxss)的完整代码如下：

```
1.  /* 游戏画布样式 */
2.  canvas {
3.    border: 1rpx solid #E64340;
4.    width: 320px;
5.    height: 320px;
6.  }
7.
8.  /* 方向键按钮整体区域 */
9.  .btnBox {
10.   display: flex;
11.   flex - direction: column;
12.   align - items: center;
13. }
14.
15. /* 方向键按钮第二行 */
16. .btnBox view {
17.   display: flex;
18.   flex - direction: row;
19. }
20.
21. /* 所有方向键按钮 */
22. .btnBox button {
23.   width: 90rpx;
24.   height: 90rpx;
25. }
26.
```

```
27. /* 所有按钮样式 */
28. button {
29.   margin: 10rpx;
30.   background-color: #E64340;
31.   color: white;
32. }
```

JS 文件(pages/game/game.js)的完整代码如下:

```
1.  //=============
2.  //游戏参数设置
3.  //=============
4.  //蛇的身长
5.  var t = 3
6.  //记录蛇的运动轨迹,用数组记录每个坐标点
7.  var snakeMap = []
8.  //蛇身单元格大小
9.  var w = 20
10. //方向代码: 上为 1、下为 2、左为 3、右为 4
11. var direction = 1
12. //蛇的初始坐标
13. var x = 0
14. var y = 0
15. //食物的初始坐标
16. var foodX = 0
17. var foodY = 0
18. //画布的宽和高
19. var width = 320
20. var height = 320
21. //游戏界面刷新的间隔时间,单位为毫秒(数字越大,蛇的速度越慢)
22. var time = 1000
23.
24. Page({
25.   /**
26.    * 页面的初始数据
27.    */
28.   data: {
29.     score: 0          //游戏当前分数
30.   },
31.
32.   /**
33.    * 自定义函数—启动游戏
34.    */
35.   gameStart: function() {
36.     //初始化游戏分数
37.     this.setData({
38.       score: 0
39.     })
40.     //初始化蛇身长度
41.     t = 3
42.     //初始化蛇身坐标
43.     snakeMap = []
44.
45.     //随机生成贪吃蛇的初始蛇头坐标
46.     x = Math.floor(Math.random() * width / w) * w
47.     y = Math.floor(Math.random() * height / w) * w
```

```
48.
49.     //随机生成蛇的初始前进方向
50.     direction = Math.ceil(Math.random() * 4)
51.
52.     //随机生成食物的初始坐标
53.     foodX = Math.floor(Math.random() * width / w) * w
54.     foodY = Math.floor(Math.random() * height / w) * w
55.
56.     //每隔 time 毫秒刷新一次游戏内容
57.     var that = this
58.     this.interval = setInterval(function() {
59.       that.gameRefresh()
60.     }, time)
61.   },
62.
63.   /**
64.    * 自定义函数—绘制食物
65.    */
66.   drawFood: function() {
67.     let ctx = this.ctx
68.     //设置食物的填充颜色
69.     ctx.setFillStyle('red')
70.     //绘制食物
71.     ctx.fillRect(foodX, foodY, w, w)
72.   },
73.
74.   /**
75.    * 自定义函数 -- 绘制贪吃蛇
76.    */
77.   drawSnake: function() {
78.     let ctx = this.ctx
79.     //设置蛇身的填充颜色
80.     ctx.setFillStyle('lightblue')
81.     //绘制全部蛇身
82.     for (var i = 0; i < snakeMap.length; i++) {
83.       ctx.fillRect(snakeMap[i].x, snakeMap[i].y, w, w)
84.     }
85.   },
86.
87.   /**
88.    * 自定义函数—碰撞检测
89.    */
90.   detectCollision: function() {
91.     //如果蛇头撞到了四周的墙壁,游戏失败
92.     if (x > width || y > height || x < 0 || y < 0) {
93.       return 1
94.     }
95.
96.     //如果蛇头撞到了蛇身,游戏失败
97.     for (var i = 0; i < snakeMap.length; i++) {
98.       if (snakeMap[i].x == x && snakeMap[i].y == y) {
99.         return 2
100.      }
101.    }
102.
```

```
103.     //没有碰撞
104.     return 0
105.   },
106.
107.   /**
108.    * 自定义函数 -- 游戏画面刷新
109.    */
110.   gameRefresh: function() {
111.     //在随机位置绘制一个食物
112.     this.drawFood()
113.
114.     //将当前坐标添加到贪吃蛇的运动轨迹坐标数组中
115.     snakeMap.push({
116.       'x': x,
117.       'y': y
118.     })
119.
120.     //数组只保留蛇身长度的数据,如果蛇前进了,则删除最旧的坐标
121.     if (snakeMap.length > t) {
122.       snakeMap.shift()
123.     }
124.
125.     //绘制贪吃蛇
126.     this.drawSnake()
127.
128.     //吃到食物的判定
129.     if (foodX == x && foodY == y) {
130.       //吃到一次食物加 10 分
131.       let score = this.data.score + 10
132.       this.setData({
133.         score: score
134.       })
135.
136.       //随机生成下一个食物的初始坐标
137.       foodX = Math.floor(Math.random() * width / w) * w
138.       foodY = Math.floor(Math.random() * height / w) * w
139.
140.       //在新的随机位置绘制食物
141.       this.drawFood()
142.
143.       //蛇身长度加 1
144.       t++
145.     }
146.
147.
148.     //在画布上绘制全部内容
149.     this.ctx.draw()
150.
151.     //根据方向移动蛇头的下一个位置
152.     switch (direction) {
153.       case 1:
154.         y -= w
155.         break
156.       case 2:
157.         y += w
```

```
158.          break
159.        case 3:
160.          x -= w
161.          break
162.        case 4:
163.          x += w
164.          break
165.      }
166.
167.      //碰撞检测,返回值为 0 表示没有撞到障碍物
168.      let code = this.detectCollision()
169.      if (code != 0) {
170.        //游戏停止
171.        clearInterval(this.interval)
172.
173.        var msg = ''
174.        if (code == 1) {
175.          msg = '失败原因:撞到了墙壁'
176.        } else if (code == 2) {
177.          msg = '失败原因:撞到了蛇身'
178.        }
179.
180.        wx.showModal({
181.          title: '游戏失败,是否重来?',
182.          content: msg,
183.          success: res => {
184.            if (res.confirm) {
185.              //重新开始游戏
186.              this.gameStart()
187.            }
188.          }
189.        })
190.      }
191.
192.    },
193.
194.    /**
195.     * 自定义函数—监听方向键:上
196.     */
197.    up: function() {
198.      direction = 1
199.    },
200.
201.    /**
202.     * 自定义函数—监听方向键:下
203.     */
204.    down: function() {
205.      direction = 2
206.    },
207.
208.    /**
209.     * 自定义函数—监听方向键:左
210.     */
211.    left: function() {
212.      direction = 3
```

```
213.   },
214.
215.   /**
216.    * 自定义函数—监听方向键：右
217.    */
218.   right: function() {
219.     direction = 4
220.   },
221.
222.   /**
223.    * 自定义函数—重新开始游戏
224.    */
225.   restartGame: function() {
226.     this.gameStart()
227.   },
228.
229.   /**
230.    * 生命周期函数—监听页面加载
231.    */
232.   onLoad: function(options) {
233.     //更新游戏刷新时间
234.     time = options.time
235.
236.     //创建画布上下文
237.     this.ctx = wx.createCanvasContext('myCanvas')
238.
239.     //开始游戏
240.     this.gameStart()
241.   },
242.
243.   /**
244.    * 生命周期函数—监听页面卸载
245.    */
246.   onUnload: function() {
247.     //游戏停止
248.     clearInterval(this.interval)
249.   },
250. })
```

提高篇



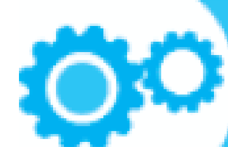
第15章 Chapter 15

小程序前端综合实例·基于模拟数据的高校新闻网

在学习了小程序的基础知识和各类 API 以后，读者不妨尝试独立动手创建一个小程序前端综合设计实例。本章将从零开始详解如何仿网易新闻小程序实现一个基于模拟数据的简易高校新闻小程序。

本章学习目标

- 综合应用所学知识创建完整的前端新闻小程序项目；
- 能够在开发过程中熟练掌握真机预览、调试等操作。



15.1　需求分析





视频讲解

本项目一共需要 3 个页面，即首页、新闻页和个人中心页，其中首页和个人中心页需要以 tabBar 的形式展示，可以点击 tab 图标互相切换。

15.1.1　首页功能需求

首页功能需求如下：

(1) 首页需要包含幻灯片播放效果和新闻列表。

(2) 幻灯片至少要有 3 幅图片自动播放。

(3) 点击新闻列表可以打开新闻全文。

15.1.2　新闻页功能需求

新闻页功能需求如下：

(1) 阅读新闻全文的页面需要显示新闻标题、图片、正文和日期。

(2) 允许点击按钮将当前阅读的新闻添加到本地收藏夹中。

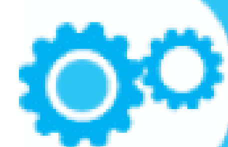
(3) 已经收藏过的新闻也可以点击按钮取消收藏。

15.1.3　个人中心页功能需求

个人中心页功能需求如下：

(1) 未登录状态下显示登录按钮，用户点击后可以显示微信头像和昵称。

(2) 登录后读取当前用户的收藏夹,展示收藏的新闻列表。
(3) 收藏夹中的新闻可以直接点击查看内容。
(4) 未登录状态下收藏夹显示为空。



15.2 项目创建



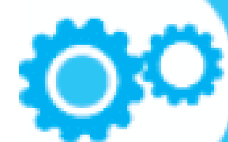


视频讲解

本项目创建选择空白文件夹 newsDemo,效果如图 15-1 所示。
单击"新建"按钮完成项目创建,然后准备手动创建页面配置文件。



图 15-1 小程序项目填写效果示意图



15.3 页面配置



15.3.1 创建页面文件



视频讲解

项目创建完毕后,在根目录中会生成文件夹 pages 用于存放页面文件。一般来说首页默认命名为 index,表示小程序运行的第一个页面;其他页面名称可以自定义。本项目需要创建 3 个页面文件,包括 index(首页页面)、detail(新闻页面)和 my(个人中心页面)。

具体操作如下:

- 将 app.json 文件 pages 属性中的 pages/logs/logs 删除;
- 在 app.json 文件 pages 属性中继续追加 pages/detail/detail 和 pages/my/my;
- 按快捷键 Ctrl+S 保存修改,然后会在 pages 文件夹下自动生成 detail 和 my 目录。

15.3.2　删除和修改文件

具体操作如下：

- 删除 utils 文件夹中的所有内容；
- 删除 pages 文件夹下的 logs 目录及其内部所有内容；
- 删除 index.wxml 和 index.wxss 中的全部代码；
- 删除 index.js 中的全部代码，并且输入关键词“page”找到第二个选项按回车让其自动补全函数，如图 15-2 所示；

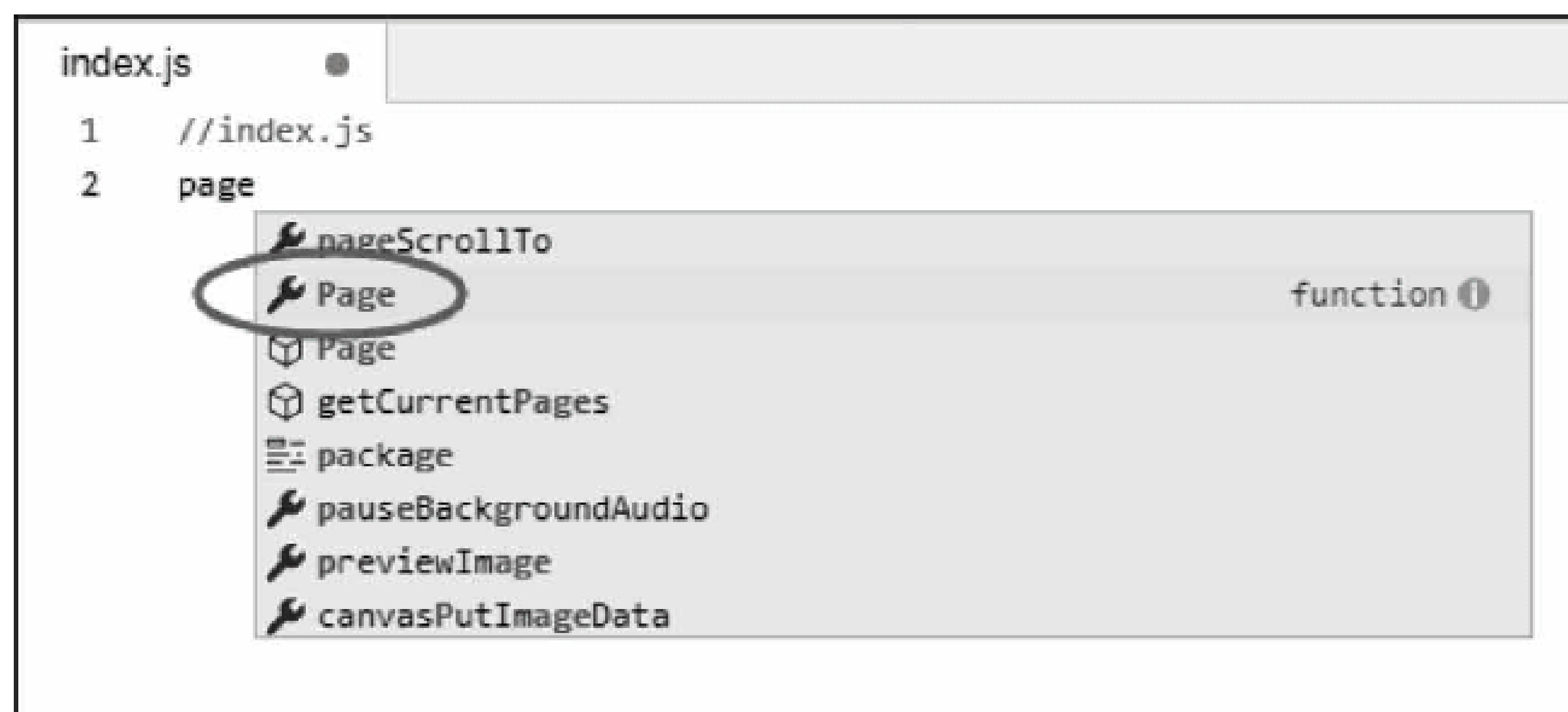


图 15-2　输入关键词创建 Page 函数

- 删除 app.wxss 中的全部代码；
- 删除 app.js 中的全部代码，并且输入关键词“app”找到第一个选项按回车让其自动补全函数，如图 15-3 所示。

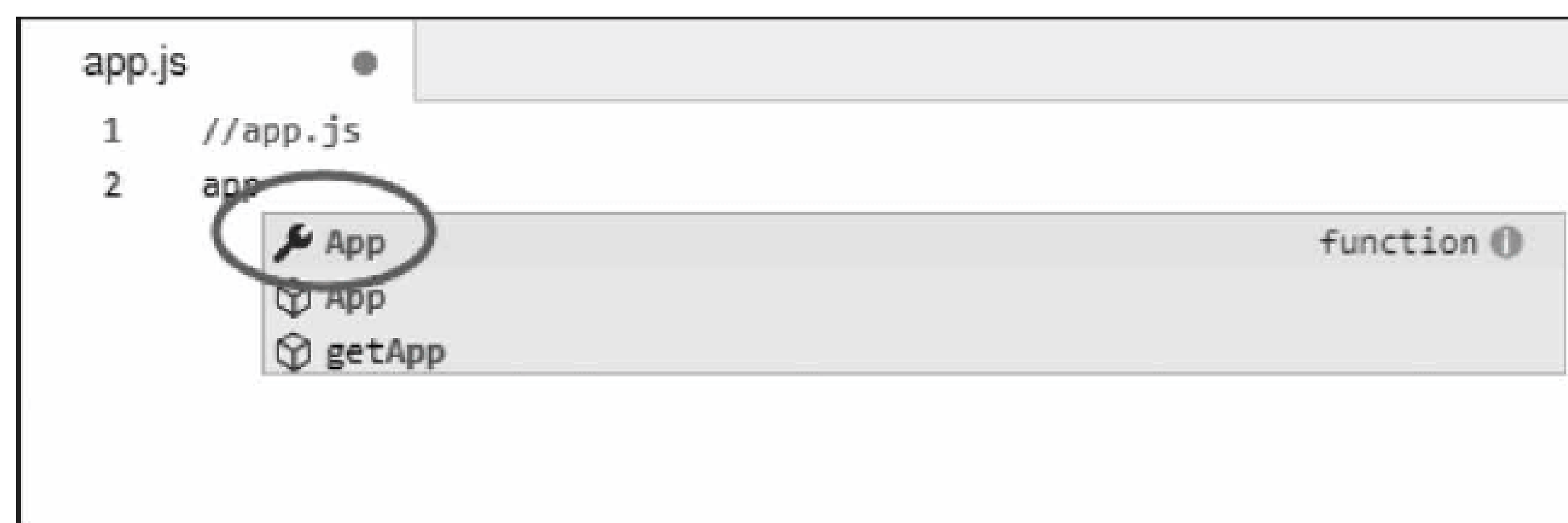


图 15-3　输入关键词创建 App 函数

15.3.3　创建其他文件

本节创建其他自定义文件，本项目还需要创建以下两个文件夹。

- images：用于存放图片素材；
- utils：用于存放公共 JS 文件。

文件夹名称由开发者自定义，创建方式与 pages 文件夹创建方式完全相同。

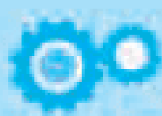
1 添加图片文件

本项目将在 tabBar 栏中用到两组图标文件，图片素材如图 15-4 所示。

右击目录结构中的 images 文件夹，选择“硬盘打开”命令，将图片复制、粘贴进去。

2 创建公共 JS 文件

右击 utils 文件夹，选择“新建”|JS 命令，输入“common”按回车键即创建公共函数 common.js，如图 15-5 所示。



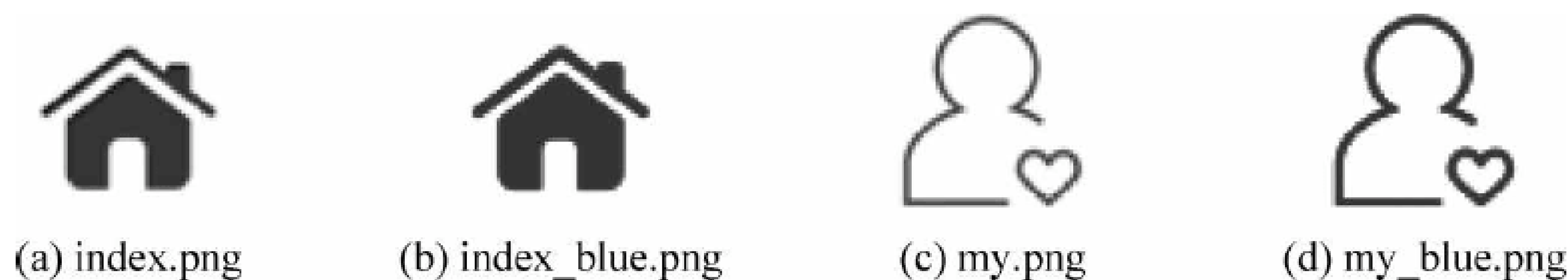


图 15-4　图标素材展示

全部完成后的目录结构如图 15-6 所示。

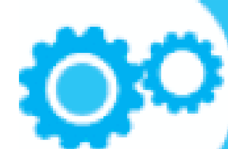


图 15-5　新建 JS 文件



图 15-6　页面文件创建完成

此时，页面文件配置全部完成，15.4 节将正式进行页面布局和样式设计。



15.4　视图设计



15.4.1　导航栏设计



视频讲解

小程序默认导航栏是黑底白字的效果，可以通过在 app.json 中对 window 属性进行重新配置来自定义导航栏效果。更改后的 app.json 文件代码如下：

```
1.  {
2.    "pages": [代码略],
3.    "window": {
4.      "navigationBarBackgroundColor": "#328EEB",
5.      "navigationBarTitleText": "我的新闻网",
6.      "navigationBarTextStyle": "white"
7.    }
8.  }
```



图 15-7 自定义导航栏效果

上述代码可以更改导航栏背景色为蓝色、字体为白色，效果如图 15-7 所示。

15.4.2 tabBar 设计


视频讲解

首先在 app.json 中追加 tarBar 的相关属性代码，更改后的 app.json 文件代码如下：

```
1.  {
2.    "pages": [代码略],
3.    "window": {代码略},
4.    "tabBar": {
5.      "color": "#000",
6.      "selectedColor": "#328EEB",
7.      "list": [
8.        {
9.          "pagePath": "pages/index/index",
10.         "iconPath": "images/index.png",
11.         "selectedIconPath": "images/index_blue.png",
12.         "text": "首页"
13.       },
14.       {
15.         "pagePath": "pages/my/my",
16.         "iconPath": "images/my.png",
17.         "selectedIconPath": "images/my_blue.png",
18.         "text": "我的"
19.       }
20.     ]
21.   }
22. }
```

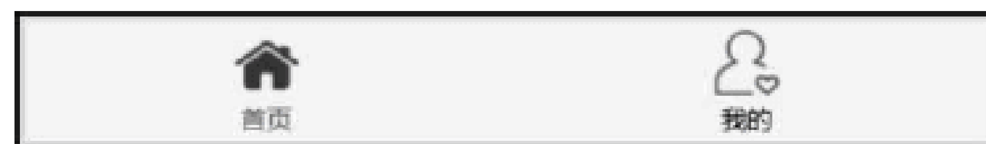


图 15-8 tabBar 完成效果图

运行效果如图 15-8 所示，此时已经可以切换首页和个人中心页了。

15.4.3 页面设计


视频讲解

1 首页设计

首页主要包含两部分内容，即幻灯片滚动和新闻列表，页面设计如图 15-9 所示。

计划使用如下组件。

- 幻灯片：<swiper>组件；
- 新闻列表：<view>容器，内部使用数组循环。

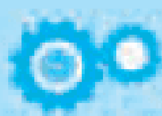
 



图 15-9 首页设计图

WXML(pages/index/index. wxml)代码如下：

```
1.  <!-- 幻灯片滚动 -->
2.  < swiper indicator - dots autoplay interval = "5000" duration = "500"> </swiper >
3.  <!-- 新闻列表 -->
4.  < view id = 'news - list'>这是新闻列表</view >
```

接着为组件添加 wx:for 属性循环显示幻灯片内容和新闻列表数据。

修改后的 WXML(pages/index/index. wxml)代码如下：

```
1.  <!-- 幻灯片滚动 -->
2.  < swiper indicator - dots = "true" autoplay = "true" interval = "5000" duration = "500">
3.    < block wx:for = "{{swiperImg}}" wx:key = 'swiper{{index}}'>
4.      < swiper - item >
5.        < image src = "{{item.src}}" class = "slide - image" />
6.      </swiper - item >
7.    </block >
8.  </swiper >
9.  <!-- 新闻列表 -->
10. < view id = 'news - list'>
11.   < view class = 'list - item' wx:for = "{{newsList}}" wx:for - item = "news" wx:key = "{{news.id}}">
12.     < image src = '{{news.poster}}'></image >
13.     < text >◇{{news.title}} -- {{news.add_date}}</text >
14.   </view >
15. </view >
```

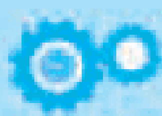
相关 WXSS(pages/index/index. wxss)代码片段如下：

```
1.  /* swiper 区域样式 */
2.  /* 1 - 1 swiper 组件 */
```

```
3.  swiper {
4.    height: 400rpx;
5.  }
6.  /* 1 - 2 swiper 中的图片 */
7.  swiper image {
8.    width: 100%;
9.    height: 100%;
10. }
11. /* 新闻列表区域样式 */
12. /* 2 - 1 新闻列表容器 */
13. #news - list {
14.   min - height: 600rpx;
15.   padding: 15rpx;
16. }
17. /* 2 - 2 列表项目 */
18. .list - item{
19.   display: flex;
20.   flex - direction: row;
21.   border - bottom: 1rpx solid gray;
22. }
23. /* 2 - 3 新闻图片 */
24. .list - item image{
25.   width:230rpx;
26.   height: 150rpx;
27.   margin: 10rpx;
28. }
29. /* 2 - 4 新闻标题 */
30. .list - item text{
31.   width: 100%;
32.   line - height: 60rpx;
33.   font - size: 10pt;
34. }
```

为了进行布局和样式效果的预览，还需要在JS文件的data中临时录入几个测试数据。相关JS(pages/index/index.js)代码片段如下：

```
1.  Page({
2.    data: {
3.      //幻灯片素材
4.      swiperImg: [
5.          { src: 'http://www.ahnu.edu.cn/__local/A/C7/68/C2C9E5E2161A466A2D54D21A63C_
DD3FEC40_4EBBB.jpg?e=.jpg'},
6.          { src: 'http://www.ahnu.edu.cn/__local/7/CC/BD/47349A7168770AC24EB5535B2AF_
35829829_6E24D.png?e=.png'},
7.          { src: 'http://www.ahnu.edu.cn/__local/2/BB/2D/9FEB9B8D0CA5E059B2C1C7E65BD_
67A0410C_3A71F.png?e=.png'}
8.      ],
9.      //临时新闻数据
10.     newsList:[{
11.       id: '264698',
12.       title: '俄罗斯联邦驻华大使杰尼索夫会见校党委书记顾家山一行并接受«力冈译文全集»赠予',
13.         poster: 'http://www.ahnu.edu.cn/__local/A/C7/68/C2C9E5E2161A466A2D54D21A63C_
DD3FEC40_4EBBB.jpg?e=.jpg',
14.       add_date: '2018 - 03 - 05'
15.     }]
```



```
16.    }
17. })
```

当前效果如图 15-10 所示。

由图可见，此时可以显示幻灯片播放和一条临时新闻。由于尚未获得新闻数据，所以暂时无法显示完整的新闻列表，仅供作为样式参考。



视频讲解

2 个人中心页设计

个人中心页主要包含两个版块，即登录面板和“我的收藏”。登录面板用于显示用户的微信头像和昵称，“我的收藏”用于显示收藏在本地的新闻列表。页面设计如图 15-11 所示。



图 15-10 首页效果图



图 15-11 个人中心页设计图

计划使用< view >组件进行整体布局，对自定义的 id 名称解释如下。

- myLogin：登录面板；
- myIcon：微信头像图片；
- nickName：微信昵称；
- myFavorites：我的收藏。

WXML(pages/my/my.wxml)代码如下：

```
1.  <!-- 登录面板 -->
2.  <view id='myLogin'></view>
3.  <!-- 我的收藏 -->
4.  <view id='myFavorites'></view>
```

接着为这两个区域添加内容，修改后的 WXML(pages/my/my.wxml)代码如下：

```
1.  <!-- 登录面板 -->
2.  <view id='myLogin'>
```

```
3.     <block>
4.       <image id='myIcon' src='{{src}}'></image>
5.       <text id='nickName'>{{nickName}}</text>
6.     </block>
7.   </view>
8.   <!-- 我的收藏 -->
9.   <view id='myFavorites'>
10.    <text>我的收藏(1)</text>
11.    <!-- 收藏的新闻列表 -->
12.    <view id='news-list'>
13.      <view class='list-item' wx:for="{{newsList}}" wx:for-item="news" wx:key=
"{{news.id}}">
14.        <image src='{{news.poster}}'></image>
15.        <text>◇{{news.title}}--{{news.add_date}}</text>
16.      </view>
17.    </view>
18.  </view>
```

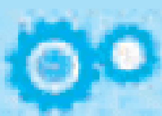
WXSS(pages/my/my.wxss)代码如下：

```
1.  /* 登录面板 */
2.  #myLogin{
3.    background-color: #328EEB;
4.    height: 400rpx;
5.    display: flex;
6.    flex-direction: column;
7.    align-items: center;
8.    justify-content: space-around;
9.  }
10. /* 1-1 头像图片 */
11. #myIcon{
12.   width: 200rpx;
13.   height: 200rpx;
14.   border-radius: 50%;
15. }
16. /* 1-2 微信昵称 */
17. #nickName{
18.   color: white;
19. }
20. /* 我的收藏 */
21. #myFavorites{
22.   padding: 20rpx;
23. }
```

由于新闻列表的样式与首页完全相同，没有必要重复样式代码，否则会造成冗余，可以将index.wxss中新闻列表样式的相关代码挪到app.wxss中公共使用。

app.wxss的代码如下：

```
1.  /* 新闻列表区域样式 */
2.  /* 2-1 新闻列表容器 */
3.  #news-list {
4.    min-height: 600rpx;
5.    padding: 15rpx;
```



```
6.  }
7.  /＊2-2 列表项目＊/
8.  .list-item{
9.    display: flex;
10.   flex-direction: row;
11.   border-bottom: 1rpx solid gray;
12. }
13. /＊2-3 新闻图片＊/
14. .list-item image{
15.   width:230rpx;
16.   height: 150rpx;
17.   margin: 10rpx;
18. }
19. /＊2-4 新闻标题＊/
20. .list-item text{
21.   width: 100%;
22.   display: block;
23.   line-height: 60rpx;
24.   font-size: 10pt;
25. }
```

为了进行布局和样式效果的预览，还需要在 JS 文件的 data 中临时录入测试数据。

JS(pages/my/my.js)代码片段如下：

```
1.  Page({
2.    data: {
3.      //临时微信用户昵称和头像
4.      nickName:'未登录',
5.      src:'/images/index.png',
6.      //临时收藏夹新闻数据
7.      newsList: [{
8.        id: '264698',
9.        title: '俄罗斯联邦驻华大使杰尼索夫会见校党委书记顾家山一行并接受«力冈译文全集»赠予',
10.        poster: 'http://www.ahnu.edu.cn/__local/A/C7/68/C2C9E5E2161A466A2D54D21A63C_
DD3FEC40_4EBBB.jpg?e=.jpg',
11.       add_date: '2018-03-05'
12.     }]
13.   }
14. })
```

当前效果如图 15-12 所示。

由图可见，此时可以显示完整的样式效果。由于尚未获得微信用户数据和收藏在本地的缓存数据，所以暂时无法显示实际内容，仅供作为样式参考。

3 新闻页设计



视频讲解

新闻页是用于给用户浏览新闻全文的，需要用户点击首页的新闻列表，然后在新窗口中打开该页面。新闻页包括新闻标题、新闻图片、新闻正文和新闻日期，页面设计如图 15-13 所示。

由于暂时没有做点击跳转的逻辑设计，所以可以在微信 web 开发者工具顶端工具栏中找到“普通编译”下拉选项，选择“添加编译模式”，然后追加对于 detail 页面的直接浏览效果，如图 15-14 所示。

此时预览就可以直接显示 detail 页面了，设计完毕后再切换回“普通编译”模式显示首页(index)即可。



图 15-12　个人中心页效果图



图 15-13　新闻页设计图



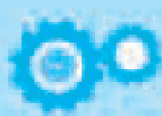
图 15-14　添加新闻页的编译模式

计划使用< view >组件进行整体布局，对自定义的 class 名称解释如下。

- container：整体容器；
- title：新闻标题区域；
- poster：新闻图片区域；
- content：新闻正文区域；
- add_date：新闻日期区域。

WXML(pages/detail/detail.wxml)代码如下：

```
1.  <view class = 'container'>
2.    <view class = 'title'>{{article.title}}</view>
3.    <view class = 'poster'>
4.      <image src = '{{article.poster}}' mode = 'widthFix'></image>
5.    </view>
6.    <view class = 'content'>{{article.content}}</view>
7.    <view class = 'add_date'>时间：{{article.add_date}}</view>
8.  </view>
```



WXSS(pages/detail/detail.wxss)代码如下：

```
1.  /*整体容器*/
2.  .container{
3.    padding: 15rpx;
4.    text-align: center;
5.  }
6.  /*新闻标题*/
7.  .title{
8.    font-size: 14pt;
9.    line-height: 80rpx;
10. }
11. /*新闻图片*/
12. .poster image{
13.   width: 100%;
14. }
15. /*新闻正文*/
16. .content{
17.   text-align: left;
18.   font-size: 12pt;
19.   line-height: 60rpx;
20. }
21. /*新闻日期*/
22. .add_date{
23.   font-size: 12pt;
24.   text-align: right;
25.   line-height: 30rpx;
26.   margin-right: 25rpx;
27.   margin-top: 20rpx;
28. }
```

为了进行布局和样式效果的预览，还需要在JS文件的data中临时录入一条测试数据。

JS(pages/detail/detail.js)代码片段如下：

```
1.  Page({
2.    data: {
3.      article:{
4.        id: '264698',
5.        title: '俄罗斯联邦驻华大使杰尼索夫会见校党委书记顾家山一行并接受«力冈译文全集»赠予',
6.        poster: 'http://www.ahnu.edu.cn/__local/A/C7/68/C2C9E5E2161A466A2D54D21A63C_DD3FEC40_4EBBB.jpg?e=.jpg',
7.        content: '本网讯(校出版社) 3月2日上午,俄罗斯驻华大使杰尼索夫在北京俄罗斯驻华大使馆会见了校党委书记顾家山,并接受了我校出版社赠予俄罗斯大使馆的十套«力冈译文全集»。俄罗斯驻华大使馆参赞梅利尼科娃、大使馆一秘伊戈尔、大使助理、塔斯社记者,我校校办主任曾黎明、出版社社长张奇才,我校杰出校友、俄罗斯人民友谊勋章和利哈乔夫院士奖获得者、中国俄罗斯文学研究会会长刘文飞教授等参加了会见.',
8.        add_date: '2018-03-05'
9.      }
10.   }
11. })
```

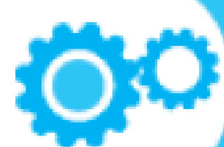


图 15-15　新闻页效果图

当前效果如图15-15所示。

由图可见,此时可以显示完整的样式效果。由于尚未获得新闻数据,所以暂时无法根据用户点击的新闻标题入口显示对应的新闻内容,仅供作为样式参考。

此时页面布局与样式设计就已完成,15.5节将介绍如何进行逻辑处理。



15.5 逻辑实现



15.5.1 公共逻辑



视频讲解

正常来说数据应该由网站群管理平台提供新闻接口,由于隐私安全、开发者条件限制等一系列问题,这里采用模拟数据进行代替。有条件的开发者可以使用第三方免费或付费新闻接口(例如聚合数据等),或自行搭建服务器提供接口。

假设已经获取到了数据,将其放在公共JS文件(utils/common.js)中,代码片段如下:

```
1.  const news = [{
2.    id: '264698',
3.    title: '俄罗斯联邦驻华大使杰尼索夫会见校党委书记顾家山一行并接受«力冈译文全集»赠予',
4.    poster: 'http://www.ahnu.edu.cn/__local/A/C7/68/C2C9E5E2161A466A2D54D21A63C_DD3FEC40_4EBBB.jpg?e=.jpg',
5.    content: '本网讯(校出版社)3月2日上午,俄罗斯驻华大使杰尼索夫在北京俄罗斯驻华大使馆会见了校党委书记顾家山,并接受了我校出版社赠予俄罗斯大使馆的十套«力冈译文全集».俄罗斯驻华大使馆参赞梅利尼科娃、大使馆一秘伊戈尔、大使助理、塔斯社记者,我校校办主任曾黎明、出版社社长张奇才,我校杰出校友、俄罗斯人民友谊勋章和利哈乔夫院士奖获得者、中国俄罗斯文学研究会会长刘文飞教授等参加了会见.',
6.    add_date: '2018-03-05'
7.  },
8.  {
9.    id: '305670',
10.   title: '我校学子在全省第八届少数民族传统体育运动会上喜获佳绩',
11.   poster: 'http://www.ahnu.edu.cn/__local/7/CC/BD/47349A7168770AC24EB5535B2AF_35829829_6E24D.png?e=.png',
12.   content: '本网讯(体育学院 邹华刚)11月18日至23日,由安徽省人民政府主办,省民委、省体育局和蚌埠市人民政府承办的安徽省第八届少数民族传统体育运动会在蚌埠市成功举行,全省16个地市代表团近1300名运动员、教练员、裁判员参加了本次运动会.\n本届运动会共设武术、民族式摔跤、毽球、蹴球、押加、高脚竞速、陀螺和板鞋竞速等8个项目.我校组建了由周慧雅、王和章、王强等23名少数民族学生组成的运动员队伍,代表芜湖市参加了高脚竞速、武术、蹴球、陀螺和板鞋竞速等5个大项的比赛.经过激烈角逐,我校运动健儿共获得4个一等奖(第1名),6个二等奖(第2~4名),9个三等奖(第5~8名)的优异战绩,出色地完成了比赛任务.',
13.   add_date: '2018-11-27'
14. },
15. {
16.   id: '304083',
17.   title: '我校学子代表国家队获中国羽毛球公开赛男双亚军',
18.   poster: 'http://www.ahnu.edu.cn/__local/2/BB/2D/9FEB9B8D0CA5E059B2C1C7E65BD_67A0410C_3A71F.png?e=.png',
19.   content: '本网讯 (体育学院 徐梦涛)11月11日,世界羽联中国羽毛球公开赛在福州落下帷幕.在男子双打半决赛中,我校2018级运动训练专业学生谭强与搭档何济霆以2:0战胜印尼组合莫哈末·阿山/亨德拉,晋级决赛.决赛中,这对年轻组合以1:2负于现世界排名第一的印度尼西亚组合,获得了本次比赛的亚军.这也是谭强在本年度内获得的最好成绩.',
20.   add_date: '2018-11-14'
21. }
```

```
22. ];
```

这里用了 3 条新闻记录作为示范，开发者可以自行添加或修改新闻内容。

接下来在 common.js 中添加自定义函数 getNewsList 和 getNewsDetail，分别用于获取新闻列表信息和指定 ID 的新闻正文内容。代码片段如下：

```
1.  //获取新闻列表
2.  function getNewsList() {
3.    let list = [];
4.    for (var i = 0; i < news.length; i++) {
5.      let obj = {};
6.      obj.id = news[i].id;
7.      obj.poster = news[i].poster;
8.      obj.add_date = news[i].add_date;
9.      obj.title = news[i].title;
10.     list.push(obj);
11.   }
12.   return list;                    //返回新闻列表
13. }
14.
15. //获取新闻内容
16. function getNewsDetail(newsID) {
17.   let msg = {
18.     code: '404',                  //没有对应的新闻
19.     news: {}
20.   };
21.   for (var i = 0; i < news.length; i++) {
22.     if (newsID == news[i].id) {   //匹配新闻 ID 编号
23.       msg.code = '200';           //成功
24.       msg.news = news[i];         //更新当前新闻内容
25.       break;
26.     }
27.   }
28.   return msg;                     //返回查找结果
29. }
```

最后需要在 common.js 中使用 module.exports 语句暴露函数出口，代码片段如下：

```
1.  module.exports = {
2.    getNewsList: getNewsList,
3.    getNewsDetail: getNewsDetail
4.  }
```

现在就完成了公共逻辑处理的部分。

然后需要在各页面的 JS 文件顶端引用公共 JS 文件，引用代码如下：

```
var common = require('../../utils/common.js')      //引用公共 JS 文件
```

需要注意这里暂时还不支持绝对路径引用，只能使用相对路径。

15.5.2 首页逻辑

首页主要有两个功能需要实现，一是展示新闻列表，二是点击新闻标题可以跳转对应的内容页面进行浏览。

1 新闻列表展示



视频讲解

新闻列表展示使用了{{newsList}},因此需要在页面 JS 文件的 onLoad 函数中获取新闻列表,并更新到 data 属性的 newsList 参数中。

相关 JS(pages/index/index.js)代码片段如下:

```
1.  Page({
2.    onLoad: function(options) {
3.      //获取新闻列表
4.      let list = common.getNewsList()
5.      //更新列表数据
6.      this.setData({newsList:list})
7.    }
8.  })
```

此时页面效果如图 15-16 所示。



图 15-16 首页新闻列表展示

2 点击跳转新闻内容



视频讲解

若希望用户点击新闻标题即可实现跳转,需要首先为新闻列表项目添加点击事件。

相关 WXML(pages/index/index.wxml)代码片段修改如下:

```
1.  <!-- 新闻列表 -->
2.  <view id = 'news - list'>
3.    <view class = 'list - item' wx:for = "{{newsList}}"
wx:for - item = "news" wx:key = "{{news.id}}">
4.      <image src = '{{news.poster}}'></image>
5.      <text bindtap = 'goToDetail' data - id = '{{news.
id}}'>◇{{news.title}} -- {{news.add_date}}</text>
6.    </view>
7.  </view>
```

具体修改为第 5 行加粗字体部分,为<text>组件添加了自定义触摸事件函数 goToDetail,并且使用 data-id 属性携带了新闻 ID 编号。

然后在对应的 detail.js 文件中添加 goToDetail 函数的内容,代码片段如下:

```
1.  Page({
2.    /**
3.     * 自定义函数 -- 跳转新页面浏览新闻内容
4.     */
5.    goToDetail: function(e) {
6.      //获取携带的 data - id 数据
7.      let id = e.currentTarget.dataset.id;
8.      //携带新闻 ID 进行页面跳转
9.      wx.navigateTo({
10.       url: '../detail/detail?id = ' + id
11.     })
12.   }
13. })
```

此时已经可以点击跳转到 detail 页面,并且成功携带了新闻 ID 数据,但是仍需在 detail 页面进行携带数据的接收处理才可显示正确的新闻内容。

15.5.3 新闻页逻辑



视频讲解

新闻页主要有两个功能需要实现，一是显示对应新闻，二是可以添加/取消新闻收藏。

1 显示对应新闻

在首页逻辑中已经实现了页面跳转并携带了新闻 ID 编号，现在需要在新闻页接收 ID 编号，并查询对应的新闻内容。

相关 JS(pages/detail/detail.js)代码片段如下：

```
1.  Page({
2.    onLoad: function(options) {
3.      //获取页面跳转来时携带的数据
4.      let id = options.id
5.      let result = common.getNewsDetail(id)
6.      //获取到新闻内容
7.      if(result.code == '200'){
8.        this.setData({article:result.news})
9.      }
10.   }
11. })
```

此时重新从首页点击新闻跳转就可以发现已经能够正确显示标题对应的新闻内容了。运行效果如图 15-17 所示。



(a) 首页新闻列表



(b) 浏览收藏的新闻

图 15-17 在首页列表中浏览新闻的效果

2 添加/取消新闻收藏

修改 detail.wxml 代码，追加两个< button >组件作为添加/取消收藏新闻的按钮，并使用

wx:if 和 wx:else 属性使其每次只存在一个。



视频讲解

WXML(pages/detail/detail.wxml)代码片段添加如下:

```
1.  <view class = 'container'>
2.    <view class = 'title'>{{article.title}}</view>
3.    <view class = 'poster'>
4.      <image src = '{{article.poster}}' mode = 'widthFix'></image>
5.    </view>
6.    <view class = 'content'>{{article.content}}</view>
7.    <view class = 'add_date'>时间: {{article.add_date}}</view>
8.    <button wx:if = '{{isAdd}}' plain bindtap = 'cancelFavorites'>♥已收藏</button>
9.    <button wx:else plain bindtap = 'addFavorites'>♥点击收藏</button>
10. </view>
```

对应的 WXSS(pages/detail/detail.wxss)代码片段添加如下:

```
1.  /*"点击收藏"按钮*/
2.  button{
3.    width: 250rpx;
4.    height: 100rpx;
5.    margin: 20rpx auto;
6.  }
```

对应 JS(pages/detail/detail.js)中的 onLoad 函数代码片段修改如下:

```
1.    onLoad: function(options) {
2.      //获取页面跳转来时携带的数据
3.      let id = options.id
4.
5.      //检查当前新闻是否在收藏夹中
6.      var article = wx.getStorageSync(id)
7.      //已存在
8.      if (article != '') {
9.        this.setData({
10.         article:article,
11.         isAdd: true
12.       })
13.     }
14.     //不存在
15.     else {
16.       let result = common.getNewsDetail(id)
17.       //获取到新闻内容
18.       if (result.code == '200') {
19.         this.setData({
20.           article: result.news,
21.           isAdd: false
22.         })
23.       }
24.     }
25.   },
```

继续在 detail.js 文件中追加 addFavorites 和 cancelFavorites 函数,用于点击添加/取消新闻收藏。对应 JS(pages/detail/detail.js)中的函数代码片段修改如下:

```
1.  Page({
2.    //添加到收藏夹
3.    addFavorites: function(options) {
```

```
4.      let article = this.data.article;              //获取当前新闻
5.      wx.setStorageSync(article.id, article);       //添加到本地缓存
6.      this.setData({ isAdd: true });                //更新按钮显示
7.    },
8.    //取消收藏
9.    cancalFavorites: function() {
10.     let article = this.data.article;              //获取当前新闻
11.     wx.removeStorageSync(article.id);             //从本地缓存删除
12.     this.setData({ isAdd: false });               //更新按钮显示
13.   },
14. })
```

现在从首页开始预览，选择其中任意一篇新闻进入 detail 页面，并尝试点击按钮收藏和取消收藏。

此时页面效果如图 15-18 所示。



(a) 已经收藏新闻　　　　(b) 取消收藏新闻



(c) 添加本地缓存

(d) 删除本地缓存

图 15-18　新闻页中“点击收藏”按钮的使用效果

15.5.4 个人中心页逻辑

个人中心页主要有 3 个功能需要实现,一是获取微信用户信息;二是获取收藏列表;三是浏览收藏的新闻。

1 获取微信用户信息



视频讲解

修改 my. wxml 代码,追加< button >组件作为登录按钮,并且使用 wx:if 和 wx:else 属性让未登录时只显示按钮,登录后只显示头像和昵称。

WXML(pages/my/my. wxml)代码片段修改如下:

```
1.  <view id='myLogin'>
2.    <block wx:if='{{isLogin}}'>
3.      <image id='myIcon' src='{{src}}'></image>
4.      <text id='nickName'>{{nickName}}</text>
5.    </block>
6.    <button wx:else>未登录,点此登录</button>
7.  </view>
```

此时页面效果如图 15-19 所示。



图 15-19 个人中心未登录状态效果图

在 my. wxml 页面修改< button >组件的代码,为其追加获取用户信息事件,代码片段如下:

```
1.   <button wx:else open-type='getUserInfo'
bindgetuserinfo='getMyInfo'>
2.     未登录,点此登录
3.   </button>
```

其中 open-type='getUserInfo'表示激活获取微信用户信息功能,然后使用 bindgetuserinfo 属性表示获得的数据将传递给自定义函数 getMyInfo,开发者也可以使用其他名称。

在 my. js 文件的 Page()内部追加 getMyInfo 函数,代码片段如下:

```
1.  getMyInfo: function(e) {
2.      console.log(e.detail.userInfo)
3.  },
```

保存后预览项目,单击按钮后如果 Console 控制台能够成功输出用户信息数据,则说明获取成功,如图 15-20 所示。

修改 my. js 文件中 getMyInfo 函数的代码,将信息更新到动态数据上,代码片段如下:

```
1.   //获取微信用户信息
2.   getMyInfo: function(e) {
3.     let info = e.detail.userInfo;
4.     this.setData({
5.       isLogin:true,                //确认登录状态
6.       src: info.avatarUrl,         //更新图片来源
7.       nickName: info.nickName      //更新昵称
8.     })
9.   },
```

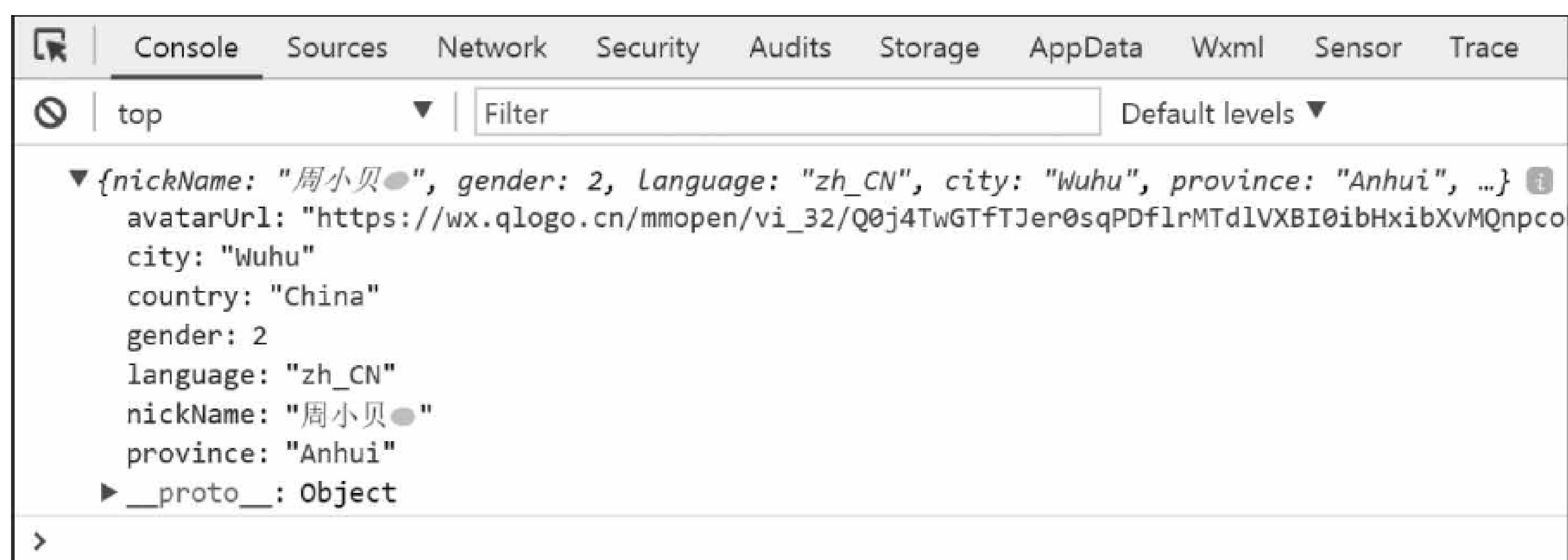


图 15-20　Console 控制台输出内容

此时就已完成登录功能，预览效果如图 15-21 所示。



(a) 初始页面效果　　　　(b) 点击按钮后的效果

图 15-21　个人中心获取微信用户信息效果图

2 获取收藏列表



视频讲解

修改 my.wxml 代码，将“我的收藏”后面的数字更改为动态数据效果。WXML(pages/my/my.wxml)代码片段如下：

```
<text>我的收藏({{num}})</text>
```

对应 JS(pages/detail/detail.js)中的 data 属性代码片段修改如下：

```
1.  Page({
2.    data: {
3.      …,
4.      num: 0
5.    }
6.  })
```

继续在 detail.js 文件中追加 getMyFavorites 函数，用于展示真正的新闻收藏列表。

对应的 JS(pages/my/my.js)代码片段如下：

```
1.  Page({
2.    //获取收藏列表
3.    getMyFavorites: function() {
4.      let info = wx.getStorageInfoSync();  //读取本地缓存信息
5.      let keys = info.keys;                //获取全部 key 信息
6.      let num = keys.length;               //获取收藏新闻数量
7.
8.      let myList = [];
9.      for (var i = 0; i < num; i++) {
10.       let obj = wx.getStorageSync(keys[i]);
11.       myList.push(obj);                  //将新闻添加到数组中
12.     }
13.     //更新收藏列表
14.     this.setData({
15.       newsList: myList,
16.       num: num
17.     });
18.   }
19. })
```

修改 my.js 中的 getMyInfo 函数，为其追加关于 getMyFavorites 函数的调用。

对应的 JS(pages/my/my.js)代码片段如下：

```
1.  Page({
2.    //获取微信用户信息
3.    getMyInfo: function(e) {
4.      let info = e.detail.userInfo;
5.      this.setData({
6.        isLogin: true,             //确认登录状态
7.        src: info.avatarUrl,       //更新图片来源
8.        nickName: info.nickName    //更新昵称
9.      })
10.     //获取收藏列表
11.     this.getMyFavorites();
12.   }
13. })
```

现在从首页开始预览，选择其中任意两篇新闻进入 detail 页面，并尝试点击收藏。然后退出切换到个人中心页，登录后查看收藏效果。

此时页面效果如图 15-22 所示。

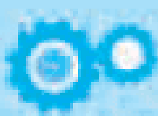


图 15-22 “我的收藏”列表效果

考虑到登录成功后用户还可以手动更改新闻的收藏状态，因此修改 my.js 中的 onShow 函数，判断如果是登录状态就刷新一下收藏列表。

对应的 JS(pages/my/my.js)代码片段如下：

```
1.  Page({
2.    /**
3.     * 生命周期函数--监听页面显示
```



```
4.      */
5.    onShow: function() {
6.      //如果已经登录
7.      if(this.data.isLogin){
8.        //更新收藏列表
9.        this.getMyFavorites()
10.     }
11.   },
12. })
```

3 浏览收藏的新闻



视频讲解

点击浏览已经收藏的新闻和首页的点击跳转新闻内容功能类似，首先修改 my.wxml 收藏列表的代码如下：

```
1.    <!-- 收藏的新闻列表 -->
2.    <view id='news-list'>
3.      <view class='list-item' wx:for="{{newsList}}" wx:for-item="news" wx:key="{{news.id}}">
4.        <image src='{{news.poster}}'></image>
5.        <text bindtap='goToDetail' data-id='{{news.id}}'>
6.           ◇{{news.title}}—{{news.add_date}}
7.        </text>
8.      </view>
```

具体修改为第 5 行加粗字体部分，为<text>组件添加了自定义触摸事件函数 goToDetail，并且使用 data-id 属性携带了新闻 ID 编号。

然后在对应的 my.js 文件中添加 goToDetail 函数的内容，代码片段如下：

```
1.  Page({
2.    goToDetail: function(e) {
3.      //获取携带的 data-id 数据
4.      let id = e.currentTarget.dataset.id;
5.      //携带新闻 ID 进行页面跳转
6.      wx.navigateTo({
7.        url: '../detail/detail?id=' + id
8.      })
9.    }
10. })
```

运行效果如图 15-23 所示。

15.5.5 清除临时数据



视频讲解

最后需要清除或注释掉一开始为了测试样式录入的临时数据，以免影响整体逻辑效果。

需要清除的数据如下。

- 首页(index.js)：data 中的临时新闻列表数据(newsList)；
- 新闻页(detail.js)：data 中的临时新闻数据(article)；
- 个人中心页(my.js)：data 中的临时收藏夹新闻数据(newsList)、临时昵称(nickName)以及临时头像路径地址(src)。

此时带有模拟新闻数据的小程序前端项目“高校新闻网”就全部完成，后续章节将为其追加后端服务器和数据库的相关内容。



(a) “我的收藏”列表



(b) 浏览收藏的新闻

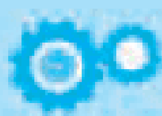
图 15-23　浏览已收藏新闻的效果

 15.6　完整代码展示 

15.6.1　应用文件代码展示

app.json 文件的完整代码如下：

```
1.  {
2.    "pages": [
3.      "pages/index/index",
4.      "pages/my/my",
5.      "pages/detail/detail"
6.    ],
7.    "window": {
8.      "navigationBarBackgroundColor": "#328EEB",
9.      "navigationBarTitleText": "我的新闻网"
10.   },
11.   "tabBar": {
12.     "color": "#000",
13.     "selectedColor": "#328EEB",
14.     "list": [
15.       {
16.         "pagePath": "pages/index/index",
17.         "text": "首页",
18.         "iconPath": "images/index.png",
19.         "selectedIconPath": "images/index_blue.png"
20.       },
```

```
21.         {
22.           "pagePath": "pages/my/my",
23.           "text": "我的",
24.           "iconPath": "images/my.png",
25.           "selectedIconPath": "images/my_blue.png"
26.         }
27.       ]
28.     }
29. }
```

app.wxss 文件的完整代码如下：

```
1.  /* 新闻列表区域样式 */
2.  /* 2-1 新闻列表容器 */
3.  #news-list {
4.    min-height: 600rpx;
5.    padding: 15rpx;
6.  }
7.  /* 2-2 列表项目 */
8.  .list-item{
9.    display: flex;
10.   flex-direction: row;
11.   border-bottom: 1rpx solid gray;
12. }
13. /* 2-3 新闻图片 */
14. .list-item image{
15.   width:230rpx;
16.   height: 150rpx;
17.   margin: 10rpx;
18. }
19. /* 2-4 新闻标题 */
20. .list-item text{
21.   width: 100%;
22.   line-height: 60rpx;
23.   font-size: 10pt;
24. }
```

15.6.2 公共函数文件代码展示

JS 文件(utils/common.js)的完整代码如下：

```
1.  //模拟新闻数据
2.  const news = [{
3.      id: '264698',
4.      title: '俄罗斯联邦驻华大使杰尼索夫会见校党委书记顾家山一行并接受«力冈译文全集»赠予',
5.        poster: 'http://www.ahnu.edu.cn/__local/A/C7/68/C2C9E5E2161A466A2D54D21A63C_DD3FEC40_4EBBB.jpg?e=.jpg',
6.      content: '本网讯(校出版社) 3 月 2 日上午,俄罗斯驻华大使杰尼索夫在北京俄罗斯驻华大使馆会见了校党委书记顾家山,并接受了我校出版社赠予俄罗斯大使馆的十套«力冈译文全集»。俄罗斯驻华大使馆参赞梅利尼科娃、大使馆一秘伊戈尔、大使助理、塔斯社记者,我校校办主任曾黎明、出版社社长张奇才,我校杰出校友、俄罗斯人民友谊勋章和利哈乔夫院士奖获得者、中国俄罗斯文学研究会会长刘文飞教授等参加了会见。',
7.      add_date: '2018-03-05'
```

```
8.    },
9.    {
10.     id: '304083',
11.     title: '我校学子代表国家队获中国羽毛球公开赛男双亚军',
12.       poster: 'http://www.ahnu.edu.cn/__local/7/CC/BD/47349A7168770AC24EB5535B2AF_
35829829_6E24D.png?e=.png',
13.     content: '本网讯 (体育学院 徐梦涛)11月11日,世界羽联中国羽毛球公开赛在福州落下帷
幕.在男子双打半决赛中,我校2018级运动训练专业学生谭强与搭档何济霆以2:0战胜印度尼西亚组
合莫哈末·阿山/亨德拉,晋级决赛.决赛中,这对年轻组合以1:2负于现世界排名第一的印度尼西亚组
合,获得了本次比赛的亚军.这也是谭强在本年度内获得的最好成绩。',
14.     add_date: '2018-11-14'
15.    },
16.    {
17.     id: '305670',
18.     title: '我校学子在全省第八届少数民族传统体育运动会上喜获佳绩',
19.       poster: 'http://www.ahnu.edu.cn/__local/2/BB/2D/9FEB9B8D0CA5E059B2C1C7E65BD_
67A0410C_3A71F.png?e=.png',
20.     content: '本网讯(体育学院 邹华刚)11月18日至23日,由安徽省人民政府主办,省民委、省
体育局和蚌埠市人民政府承办的安徽省第八届少数民族传统体育运动会在蚌埠市成功举行,全省16个
地市代表团近1300名运动员、教练员、裁判员参加了本次运动会.\n本届运动会共设武术、民族式摔跤、
毽球、蹴球、押加、高脚竞速、陀螺和板鞋竞速等8个项目.我校组建了由周慧雅、王和章、王强等23名少
数民族学生组成的运动员队伍,代表芜湖市参加了高脚竞速、武术、蹴球、陀螺和板鞋竞速等5个大项的
比赛.经过激烈角逐,我校运动健儿共获得4个一等奖(第1名),6个二等奖(第2~4名),9个三等奖
(第5~8名)的优异战绩,出色地完成了比赛任务。',
21.     add_date: '2018-11-27'
22.    }
23.  ];
24.
25.  //获取新闻列表
26.  function getNewsList() {
27.    let list = [];
28.    for (var i = 0; i < news.length; i++) {
29.      let obj = {};
30.      obj.id = news[i].id;
31.      obj.poster = news[i].poster;
32.      obj.add_date = news[i].add_date;
33.      obj.title = news[i].title;
34.      list.push(obj);
35.    }
36.    return list;                    //返回新闻列表
37.  }
38.
39.  //获取新闻内容
40.  function getNewsDetail(newsID) {
41.    let msg = {
42.      code: '404',                  //没有对应的新闻
43.      news: {}
44.    };
45.    for (var i = 0; i < news.length; i++) {
46.      if (newsID == news[i].id) {   //匹配新闻 ID 编号
47.        msg.code = '200';           //成功
48.        msg.news = news[i];         //更新当前新闻内容
```

```
49.         break;
50.       }
51.     }
52.     return msg;                    //返回查找结果
53.   }
54.
55.   /*
56.    * 对外暴露接口
57.    */
58.   module.exports = {
59.     getNewsList: getNewsList,
60.     getNewsDetail: getNewsDetail
61.   }
```

15.6.3 页面文件代码展示

1 首页代码展示

WXML 文件(pages/index/index.wxml)的完整代码如下：

```
1.  <!-- 幻灯片 -->
2.  <swiper indicator-dots autoplay interval='5000' duration='500'>
3.    <block wx:for='{{[newsList[0],newsList[1],newsList[2]]}}' wx:key='swiper{{index}}'>
4.      <swiper-item>
5.        <image src='{{item.poster}}'></image>
6.      </swiper-item>
7.    </block>
8.  </swiper>
9.
10. <!-- 新闻列表 -->
11. <view id='news-list'>
12.   <view class='list-item' wx:for='{{newsList}}' wx:key='news{{index}}' wx:for-item=
'news'>
13.     <image src='{{news.poster}}'></image>
14.     <text bindtap='goToDetail' data-id='{{news.id}}'>
          ◇{{news.title}} -- {{news.add_date}}</text>
15.   </view>
16. </view>
```

WXSS 文件(pages/index/index.wxss)的完整代码如下：

```
1.  /* swiper 区域样式 */
2.  /* 1-1 swiper 组件 */
3.  swiper {
4.    height: 400rpx;
5.  }
6.  /* 1-2 swiper 中的图片 */
7.  swiper image {
8.    width: 100%;
9.    height: 100%;
10. }
```

JS 文件(pages/index/index.js)的完整代码如下：

```
1.  var common = require('../../utils/common.js')      //引用公共 JS 文件
```

```
2.  Page({
3.    data: {
4.      //幻灯片素材
5.      swiperImg: [
6.          { src: 'http://www.ahnu.edu.cn/__local/A/C7/68/C2C9E5E2161A466A2D54D21A63C_
DD3FEC40_4EBBB.jpg?e=.jpg' },
7.          { src: 'http://www.ahnu.edu.cn/__local/7/CC/BD/47349A7168770AC24EB5535B2AF_
35829829_6E24D.png?e=.png'},
8.          { src: 'http://www.ahnu.edu.cn/__local/2/BB/2D/9FEB9B8D0CA5E059B2C1C7E65BD_
67A0410C_3A71F.png?e=.png'}
9.      ]
10.   },
11.   goToDetail: function(e) {
12.     //获取携带的 data-id 数据
13.     let id = e.currentTarget.dataset.id;
14.     //携带新闻 ID 进行页面跳转
15.     wx.navigateTo({
16.       url: '../detail/detail?id=' + id
17.     })
18.   },
19.   onLoad: function(options) {
20.     //获取新闻列表
21.     let list = common.getNewsList()
22.     //更新列表数据
23.      this.setData({ newsList: list })
24.   }
25. })
```

2 新闻页代码展示

WXML 文件(pages/detail/detail.wxml)的完整代码如下：

```
1.  <view class='container' wx:if='{{article.id}}'>
2.    <view class='title'>{{article.title}}</view>
3.    <view class='poster'>
4.      <image src='{{article.poster}}' mode='widthFix'></image>
5.    </view>
6.    <view class='content'>{{article.content}}</view>
7.    <view class='add_date'>时间：{{article.add_date}}</view>
8.    <button wx:if='{{isAdd}}' plain bindtap='cancelFavorites'>♥已收藏</button>
9.    <button wx:else plain bindtap='addFavorites'>♥点击收藏</button>
10. </view>
```

WXSS 文件(pages/detail/detail.wxss)的完整代码如下：

```
1.  /*整体容器*/
2.  .container{
3.    padding: 15rpx;
4.    text-align: center;
5.  }
6.  /*新闻标题*/
7.  .title{
8.    font-size: 14pt;
9.    line-height: 80rpx;
10. }
11. /*新闻图片*/
12. .poster image{
```



```
13.   width: 100 % ;
14. }
15. / * 新闻正文 * /
16. .content{
17.   text - align: left;
18.   font - size: 12pt;
19.   line - height: 60rpx;
20. }
21. / * 新闻日期 * /
22. .add_date{
23.   font - size: 12pt;
24.   text - align: right;
25.   line - height: 30rpx;
26.   margin - right: 25rpx;
27.   margin - top: 20rpx;
28. }
29. / * "点击收藏"按钮 * /
30. button{
31.   width: 250rpx;
32.   height: 100rpx;
33.   margin: 20rpx auto;
34. }
```

JS 文件(pages/detail/detail.js)的完整代码如下：

```
1.  var common = require('../../utils/common.js')          //引用公共 JS 文件
2.  Page({
3.    //添加到收藏夹
4.    addFavorites: function(options) {
5.      let article = this.data.article;                    //获取当前新闻
6.      wx.setStorageSync(article.id, article);             //添加到本地缓存
7.      this.setData({ isAdd: true });                      //更新按钮显示
8.    },
9.    //取消收藏
10.   cancelFavorites: function() {
11.     let article = this.data.article;                    //获取当前新闻
12.     wx.removeStorageSync(article.id);                   //从本地缓存删除
13.     this.setData({ isAdd: false });                     //更新按钮显示
14.   },
15.   onLoad: function(options) {
16.     //获取页面跳转来时携带的数据
17.     let id = options.id
18.     //检查当前新闻是否在收藏夹中
19.     var article = wx.getStorageSync(id)
20.     //已存在
21.     if (article != '') {
22.       this.setData({
23.         article:article,
24.         isAdd: true
25.       })
26.     }
27.     //不存在
28.     else {
29.       let result = common.getNewsDetail(id)
30.       //获取到新闻内容
31.       if (result.code == '200') {
```

```
32.          this.setData({
33.            article: result.news,
34.            isAdd: false
35.          })
36.        }
37.      }
38.    }
39.  })
```

3 个人中心页代码展示

WXML 文件(pages/my/my.wxml)的完整代码如下：

```
1.  <!-- 登录面板 -->
2.  <view id='myLogin'>
3.    <block wx:if='{{isLogin}}'>
4.      <image id='myIcon' src='{{src}}'></image>
5.      <text id='nickName'>{{nickName}}</text>
6.    </block>
7.    <button wx:else open-type='getUserInfo' bindgetuserinfo='getMyInfo'>
8.      未登录,点此登录
9.    </button>
10. </view>
11. <!-- 我的收藏 -->
12. <view id='myFavorites'>
13.   <text>我的收藏({{num}})</text>
14.   <!-- 收藏的新闻列表 -->
15.   <view id='news-list'>
16.     <view class='list-item' wx:for="{{newsList}}" wx:for-item="news" wx:key=
"{{news.id}}">
17.       <image src='{{news.poster}}'></image>
18.       <text class='news-title' bindtap='goToDetail' data-id='{{news.id}}'>
19.         ◇{{news.title}} -- {{news.add_date}}
20.       </text>
21.     </view>
22.   </view>
23. </view>
```

WXSS 文件(pages/my/my.wxss)的完整代码如下：

```
1.  /* 登录面板 */
2.  #myLogin{
3.    background-color: #328EEB;
4.    height: 400rpx;
5.    display: flex;
6.    flex-direction: column;
7.    align-items: center;
8.    justify-content: space-around;
9.  }
10. /* 1-1 头像图片 */
11. #myIcon{
12.   width: 200rpx;
13.   height: 200rpx;
14.   border-radius: 50%;
15. }
16. /* 1-2 微信昵称 */
17. #nickName{
```

```
18.     color: white;
19.   }
20.   /* 我的收藏 */
21.   #myFavorites{
22.     padding: 20rpx;
23.   }
```

JS 文件(pages/my/my.js)的完整代码如下：

```
1.   Page({
2.     data: {
3.       num: 0
4.     },
5.     //获取微信用户信息
6.     getMyInfo: function(e) {
7.       let info = e.detail.userInfo;
8.       this.setData({
9.         isLogin: true,                              //确认登录状态
10.        src: info.avatarUrl,                        //更新图片来源
11.        nickName: info.nickName                     //更新昵称
12.      })
13.      //获取收藏列表
14.      this.getMyFavorites();
15.    },
16.    //获取收藏列表
17.    getMyFavorites: function() {
18.      let info = wx.getStorageInfoSync();          //读取本地缓存信息
19.      let keys = info.keys;                        //获取全部 key 信息
20.      let num = keys.length;                       //获取收藏新闻数量
21.
22.      let myList = [];
23.      for (var i = 0; i < num; i++) {
24.        let obj = wx.getStorageSync(keys[i]);
25.        myList.push(obj);                          //将新闻添加到数组中
26.      }
27.      //更新收藏列表
28.      this.setData({
29.        newsList: myList,
30.        num: num
31.      });
32.    },
33.    goToDetail: function(e) {
34.      //获取携带的 data-id 数据
35.      let id = e.currentTarget.dataset.id;
36.      //携带新闻 ID 进行页面跳转
37.      wx.navigateTo({
38.        url: '../detail/detail?id=' + id
39.      })
40.    }
41.    /**
42.     * 生命周期函数--监听页面显示
43.     */
44.    onShow: function() {
45.      //如果已经登录
46.      if(this.data.isLogin){
47.        //更新收藏列表
```

```
48.         this.getMyFavorites()
49.       }
50.     },
51.  })
```



15.7 项目小结



通过基于模拟数据的高校新闻网项目的开发练习，主要综合复习了以下知识点和操作：

(1) 小程序项目的创建步骤。

(2) 手动新建小程序应用文件和页面文件的步骤。

(3) 导航栏的标题、背景颜色和文字颜色的设置方法。

(4) tabBar 的创建方法。

(5) 页面布局和样式设计的基本方法。

(6) 使用双花括号{{}}生成动态数据的方法。

(7) 使用 setData 重置动态数据的方法。

(8) wx:for、wx:if 和 wx:else 属性的用法。

(9) 公共函数的创建、导出和调用方法。

(10) 本地缓存数据的读取和删除。

(11) 使用按钮的 open-type 属性读取微信个人信息的方法。

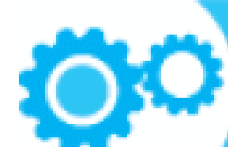
(12) 页面导航跳转和数据携带的用法。

(13) 项目预览和真机调试的步骤。



第16章 Chapter 16 小程序全栈开发·基于WAMP的高校新闻网

小程序前端项目对接上后端服务器、数据库和存储可以称为小程序的全栈开发。本章将以 WAMP(Windows+Apache+MySQL+PHP)组合为例从零开始详解如何部署服务器、搭建数据库以及实现后端接口，基于前面第 15 章的高校新闻网小程序项目进行全栈开发改造。



本章学习目标

- 综合应用所学知识创建完整的新闻小程序项目；
- 熟练掌握服务器部署、数据库搭建和接口的实现。

16.1 初始化项目



16.1.1 现有项目导入



视频讲解

本项目是基于第 15 章 newsDemo 项目的改造，因此无须新建空白项目，直接复制第 15 章项目文件夹到新的路径并且重命名为 newsDemo_WAMP（该名称由开发者自定义）即可导入微信 web 开发者工具中进行后续的开发。

注意导入时修改一下项目名称，因为开发者工具中不允许有同名项目存在。

效果如图 16-1 所示。

单击“导入”按钮完成项目导入，即可进行后续的改造工作了。

16.1.2 后端逻辑实现

后端逻辑实现包含服务器部署、数据库搭建以及接口的实现，本章以 Windows+Apache+MySQL+PHP(简称为 WAMP)服务器环境为例进行示范。小程序对于接口制作所使用的数据库、后端语言和框架均没有固定要求，这里仅是其中一种示例，开发者完全可以根据自己所擅长的领域自由选择。

1 服务器部署



视频讲解



视频讲解

请根据“附录 A 服务器部署”中的内容进行服务器的搭建和部署工作。

2 数据库搭建

请根据“附录 B 可视化数据库搭建”中的内容进行数据库和



图 16-1 小程序项目填写效果示意图

数据表的搭建工作。

例如创建一个名称为 news 的数据库，并在其中创建名称为 campus_news 的数据表。campus_news 表的结构参考表 16-1。

表 16-1 campus_news 表结构

字段名称	字段类型	字段长度	字段含义	主键
id	varchar	10	新闻编号	是
title	varchar	120	新闻标题	否
poster	varchar	256	新闻图片 URL	否
content	longtext	1000	新闻内容	否
add_date	date	10	添加日期	否

其中字段名称、类型、长度等均可以由开发者按照实际情况进行自定义。

然后根据实际情况录入若干条新闻数据，如图 16-2 所示。

3 接口的实现



视频讲解

请根据“附录 C 后端框架搭建”学习框架部署、服务器对接以及接口制作。

首先自定义函数 getNewsList 和 getNewsById，分别用于获取新闻列表和根据 ID 编号获取指定的新闻内容：

```
1.  <?php
2.  namespace Home\Controller;
3.  use Think\Controller;
4.
5.  class IndexController extends Controller
6.  {
7.      public function getNewsList(){ }            //获取新闻列表
```

campus_news @news (测试) - 表

文件 编辑 查看 窗口 帮助

导入向导 导出向导 筛选向导 网格查看 表单查看 备注 十六进制 图像 升序排序

id	title	poster	content	add_date
288255	“文化名家进高校”专场文艺演出首站走进安	http://www.ahnu.edu.cn/u	本网讯（大学生记者团 肖	2018-06-14
288276	省文联“文化名家进高校”活动走进安徽师范	http://www.ahnu.edu.cn/u	本网讯（美术学院 王伟）	2018-06-14
288377	我校外国语学院教师参与制定中国英语能力等	http://www.ahnu.edu.cn/u	本网讯 近日我校外国语学	2018-06-14
288399	学校深入学习贯彻习近平总书记在北京大学师	http://www.ahnu.edu.cn/u	本网讯（组织部 廖仲明）	2018-06-14
288431	学校专题部署毕业生文明离校工作	http://www.ahnu.edu.cn/u	本网讯（学生处 王亚男/	2018-06-15
288499	我校骨干教师第四期赴俄培训团举行行前培训	http://www.ahnu.edu.cn/u	本网讯（国际交流合作处	2018-06-15
288505	【情系母校】郭锐评、陈礼勤：漂洋过海尽 ‘	http://www.ahnu.edu.cn/u	正值安徽师范大学90华诞	2018-06-15
288530	安徽师范大学与安徽省高分卫星数据应用中心	http://kyc.ahnu.edu.cn/up	本网讯（科研处 王静）6月	2018-06-15
288550	学校举行安徽师范大学首届民族文化节	http://www.ahnu.edu.cn/u	本网讯（学工处 学生助理	2018-06-15
288553	“文化名家进高校”活动走进安徽师范大学文	http://www.ahnu.edu.cn/u	本网讯（文学院 魏晓涵	2018-06-15
288856	安徽师范大学后勤保障：从“汗水”走向“智	http://www.ahnu.edu.cn/u	“我们寝室的桌子坏了，i	2018-06-20
288866	我校乒乓球队代表中国参加国际乒联朝鲜公开	http://www.ahnu.edu.cn/u	本网讯（体育学院 新闻中	2018-06-20
289038	我校举行客座教授聘任暨雷忠平奖助学金签约	http://www.ahnu.edu.cn/u	本网讯（地旅学院 王美）	2018-06-21
289154	我校学子在省大学生“创青春”创业大赛中喜	http://www.ahnu.edu.cn/u	青通社讯（汪凯 甘伟）近	2018-06-22
289240	第二届语言教学与研究国际学术研讨会在我校	http://wxy.ahnu.edu.cn/up	本网讯（文学院）鼓瑟吹	2018-06-25
289337	我校首届教职工气排球比赛顺利举行	http://www.ahnu.edu.cn/u	本网讯（文/校工会李平 |	2018-06-25
289511	【青春毕业季】安徽师范大学隆重举行2018届	http://www.ahnu.edu.cn/u	本网讯（校大学生记者团	2018-06-27
301986	【运动员专访】跑道上的梦想：记勇夺省运会	http://www.ahnu.edu.cn/u	本网讯（外国语学院 李	2018-10-29
302308	我校在全国首届“图书馆杯主题海报创意设计	http://www.ahnu.edu.cn/u	本网讯（图书馆)近日，由	2018-10-31

SELECT * FROM `campus_news` LI　　第 1 条记录 (共 19 条) 于 1 页

图 16-2　campus_news 表录入数据参考图

```
8.      public function getNewsById(){ }          //根据 ID 编号获取新闻内容
9.  }
```

获取新闻列表的接口的详细代码如下：

```
1.     public function getNewsList()
2.     {
3.        $p = I('page');                            //获取当前页数
4.        $page = ($p!= '') ? intval($p) : 1;         //当前页数取整
5.        $rows = 5;                                  //每次获取的新闻数量
6.
7.        $offset = ($page - 1) * $rows;              //计算从第几条数据开始获取
8.        $m = M();                                   //创建查询模型
9.        $count = $m->query('select count(id) n from campus_news');   //查询新闻总数
10.       $rs = $m->query('select id,title,add_date,poster from campus_news limit '.$offset.','.
$rows);                                               //查询新闻列表数据
11.       $msg['total'] = $count[0]['n'];             //获取新闻总数
12.       $msg['list'] = $rs;                         //获取当前页的新闻列表数据
13.
14.      echo json_encode($msg);                      //发送 JSON 数据
15.    }
```

根据 ID 编号获取指定新闻内容的接口的详细代码如下：

```
1.     public function getNewsById()
2.     {
3.        $id = I('id');                              //获取新闻 ID 编号
```

```
4.      $m = M();                              //创建查询模型
5.      $rs = $m->query('select * from campus_news where id='.$id);    //查询新闻内容
6.      echo json_encode($rs);                 //发送 JSON 数据
7.    }
```

此时接口就制作完成了，开发者可以先使用浏览器进行访问以检测数据是否获得。

16.1.3 公共逻辑



视频讲解

在公共 JS 文件(utils/common.js)中删除原先的所有代码，并进行服务器接口的地址配置。

代码片段如下：

```
1.  //服务器域名地址
2.  var baseUrl = 'http:                //127.0.0.1/myNews/'
3.
4.  //获取新闻列表接口
5.  var getNewsList = baseUrl + 'Index/getNewsList'
6.
7.  //根据新闻 ID 获取新闻内容接口
8.  var getNewsById = baseUrl + 'Index/getNewsById'
```

注意，这里的 127.0.0.1 是以本机模拟服务器效果为例，开发者如果有条件搭建自己的服务器，可以将此处替换成实际服务器的 IP 地址或域名地址。

接下来继续在 common.js 中添加自定义函数 goToDetail，用于打开新闻正文页面，并携带新闻 ID 编号，代码片段如下：

```
1.  /**
2.    * 自定义函数 -- 跳转新闻浏览页面
3.    */
4.  function goToDetail(id) {
5.    wx.navigateTo({
6.      url: '../detail/detail?id=' + id,
7.    })
8.  }
```

最后需要在 common.js 中使用 module.exports 语句暴露变量与函数出口，代码片段如下：

```
1.  module.exports = {
2.    getNewsList: getNewsList,
3.    getNewsById: getNewsById,
4.    goToDetail: goToDetail
5.  }
```

现在就完成了公共逻辑处理的部分。

然后需要在各页面的 JS 文件顶端引用公共 JS 文件，引用代码如下：

```
var common = require('../../utils/common.js')        //引用公共 JS 文件
```

需要注意小程序在这里暂时还不支持绝对路径引用，只能使用相对路径。



16.2 首页改造



16.2.1 新闻列表展示



首先在页面底部添加一个“加载更多”按钮，以备后续使用。

修改后的 WXML 文件（pages/index/index.wxml）代码片段如下：

```
1.  <!-- 新闻列表 -->
2.  <view id='news-list'>
3.    <view class='list-item' wx:for="{{newsList}}" wx:for-item="news" wx:key="{{news.
id}}">
4.      <image src='{{news.poster}}'></image>
5.      <text>◇{{news.title}}——{{news.add_date}}</text>
6.    </view>
7.    <button plain loading="{{loading}}">
8.      {{loadMoreText}}
9.    </button>
10. </view>
```

然后在 JS 文件的 data 中添加按钮的状态和文字描述。

有关 JS 文件（pages/index/index.js）代码片段如下：

```
1.  Page({
2.    /**
3.     * 页面的初始数据
4.     */
5.    data: {
6.      loading: false,
7.      loadMoreText: '加载更多'
8.    },
9.  })
```

在 JS 文件顶端定义公共变量 isEnd 和 currentPage，分别用于表示新闻是否全部加载和当前处于第几页，有关 JS 文件（pages/index/index.js）代码片段如下：

```
1.  var isEnd = false
2.  var currentPage = 1
3.  Page({
4.    …
5.  })
```

创建自定义函数 getNewsListByPage 用于向服务器发送请求，获取新闻总数和指定页数上的新闻列表，并使用 setData 函数渲染到页面上。

有关 JS 文件（pages/index/index.js）代码片段如下：

```
1.  Page({
2.    /**
3.     * 自定义函数 -- 获取指定页数的新闻列表
4.     */
5.    getNewsListByPage: function(page) {
6.      var that = this
7.      wx.request({
```

```
8.          url: common.getNewsList,
9.          data: {
10.           page: page
11.         },
12.         success: function(res) {
13.           //获取新闻总数
14.           let total = res.data.total
15.           //追加更多新闻数据
16.           let list = that.data.newsList.concat(res.data.list)
17.           //更新新闻数据和新闻总数
18.           that.setData({
19.             newsList: list,
20.             total: total
21.           })
22.           //如果已经显示全部新闻
23.           if (list.length == total) {
24.             isEnd = true
25.             that.setData({
26.               loadMoreText: '已无更多'
27.             })
28.           } else {
29.             //翻到下一页
30.             currentPage++
31.           }
32.         }
33.       })
34.     },
35. })
```

在 onLoad 函数中调用 getNewsListByPage 获取第一页新闻数据。

有关 JS 文件(pages/index/index.js)代码片段如下：

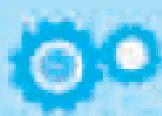
```
1.  Page({
2.    /**
3.     * 生命周期函数--监听页面加载
4.     */
5.    onLoad: function(options) {
6.      this.getNewsListByPage(1)                    //获取第一页新闻数据
7.    },
8.  })
```

修改幻灯片滚动区域的循环数组，将其更新为新闻列表中的前 3 条记录。

有关 WXML 文件(pages/index/index.wxml)代码片段修改如下：

```
1.  <!-- 幻灯片 -->
2.  <swiper indicator-dots autoplay interval='5000' duration='500'>
3.    <block wx:for='{{[newsList[0],newsList[1],newsList[2]]}}' wx:key='swiper{{index}}'>
4.      <swiper-item>
5.        <image src='{{item.poster}}'></image>
6.      </swiper-item>
7.    </block>
8.  </swiper>
```

此时页面效果如图 16-3 所示。





(a) 页面初始效果



(b) 页面底部效果

图 16-3 首页新闻列表展示

16.2.2 加载更多新闻



视频讲解

为首页新闻列表底部的“加载更多”按钮追加 bindtap 属性绑定点击事件。有关 WXML 文件(pages/index/index.wxml)代码片段修改如下：

```
1.  <!-- 新闻列表 -->
2.  <view id = 'news - list'>
3.    <view class = 'list - item' wx:for = "{{newsList}}" wx:for - item = "news" wx:key = "{{news.id}}">
4.      <image src = '{{news.poster}}'></image>
5.      <text>◇{{news.title}} -- {{news.add_date}}</text>
6.    </view>
7.    <button plain loading = "{{loading}}" bindtap = "loadMore">
8.      {{loadMoreText}}
9.    </button>
10. </view>
```

在 JS 文件中创建 loadMore 函数用于加载当前页面新闻，当用户点击时尝试向服务器要求获取下一页新闻列表数据，当没有更多数据时显示“已无更多”字样，并禁止点击。

有关 JS 文件(pages/index/index.js)代码片段如下：

```
1.  Page({
2.    /**
3.     * 自定义函数 -- 加载更多新闻
4.     */
```

```
5.     loadMore: function() {
6.       //如果新闻尚未全部加载完毕,并且按钮不处于正在加载状态
7.       if (!isEnd && !this.data.loading) {
8.         //让按钮出现加载动画
9.         this.setData({
10.          loading: true
11.        })
12.        setTimeout(() => {
13.          //加载当前页面新闻数据
14.          this.getNewsListByPage(currentPage)
15.          //停止按钮加载动画
16.          this.setData({
17.            loading: false
18.          })
19.        }, 1000)
20.      }
21.    },
22.  })
```

此时页面效果如图 16-4 所示。



(a) 页面初始底部效果



(b) 首次加载后底部效果



(c) 全部加载完毕后底部效果

图 16-4 首页新闻列表加载效果展示

16.2.3 点击跳转新闻内容



视频讲解

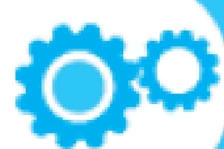
在对应的 detail.js 文件中修改 goToDetail 函数的内容,代码片段如下:

```
1.  Page({
2.    /**
3.     * 自定义函数 -- 跳转新闻浏览页面
4.     */
```

```
5.     goToDetail: function(e) {
6.       let id = e.currentTarget.dataset.id
7.       common.goToDetail(id)
8.     },
9.   })
```

此时已经可以点击跳转到 detail 页面,并且成功携带了新闻 ID 数据,但是仍需在 detail 页面进行携带数据的接收处理才可显示正确的新闻内容。



16.3 新闻页改造





视频讲解

新闻页的改造主要集中在如何获取新闻正文内容,在首页逻辑中已经实现了页面跳转并携带了新闻 ID 编号,现在需要在新闻页面接收 ID 编号,并查询对应的新闻内容。

有关 JS 文件(pages/detail/detail.js)代码片段修改如下:

```
1.  Page({
2.    /**
3.     * 生命周期函数--监听页面加载
4.     */
5.    onLoad: function(options) {
6.      //获取携带的新闻 ID 编号
7.      let id = options.id
8.      //检查当前新闻是否在收藏夹中
9.      var article = wx.getStorageSync(id)
10.     //已存在
11.     if (article != '') {
12.       this.setData({
13.         isAdd: true,
14.         article: article
15.       })
16.     }
17.     //不存在
18.     else {
19.       this.setData({ isAdd: false })
20.       var that = this
21.       //向服务器发出请求获取新闻
22.       wx.request({
23.         url: common.getNewsById,
24.         data: {
25.           id: id
26.         },
27.         success: function(res) {
28.           that.setData({ article: res.data[0] })
29.         }
30.       })
31.     }
32.   },
33. })
```

此时重新从首页点击新闻跳转就可以发现已经能够正确显示标题对应的新闻内容了。运行效果如图 16-5 所示。



(a) 首页新闻列表



(b) 点击浏览新闻

图 16-5 在首页新闻列表中浏览新闻的效果

 16.4 个人中心页改造 


视频讲解

个人中心页的改造主要集中在如何浏览已收藏的新闻，与首页的点击跳转新闻内容功能类似，在对应的 my.js 文件中修改 goToDetail 函数的内容，代码片段如下：

```
1.  Page({
2.    /**
3.     * 自定义函数 -- 跳转新闻浏览页面
4.     */
5.    goToDetail: function(e) {
6.      let id = e.currentTarget.dataset.id
7.      common.goToDetail(id)
8.    },
9.  })
```

最终运行效果如图 16-6 所示。

此时小程序全栈开发的高校新闻网项目就正式改造完毕，开发者也可以根据自己所擅长的技术方向重新选择其他服务器或后端开发语言，例如 Java、Node.js 等均可实现该项目。



(a) 我的收藏列表



(b) 浏览收藏的新闻

图 16-6 浏览已收藏新闻的效果

 16.5 完整代码展示 

16.5.1 应用文件代码展示

app.json 文件的完整代码如下：

```
1.  {
2.    "pages": [
3.      "pages/index/index",
4.      "pages/my/my",
5.      "pages/detail/detail"
6.    ],
7.    "window": {
8.      "navigationBarBackgroundColor": "#328EEB",
9.      "navigationBarTitleText": "我的新闻网"
10.   },
11.   "tabBar": {
12.     "color": "#000",
13.     "selectedColor": "#328EEB",
14.     "list": [
15.       {
16.         "pagePath": "pages/index/index",
17.         "text": "首页",
18.         "iconPath": "images/index.png",
19.         "selectedIconPath": "images/index_blue.png"
20.       },
```

```
21.         {
22.           "pagePath": "pages/my/my",
23.           "text": "我的",
24.           "iconPath": "images/my.png",
25.           "selectedIconPath": "images/my_blue.png"
26.         }
27.       ]
28.     }
29.  }
```

app.wxss 文件的完整代码如下：

```
1.  /* 新闻列表区域样式 */
2.  /* 2-1 新闻列表容器 */
3.  #news-list{
4.    min-height: 600rpx;
5.    padding: 15rpx;
6.  }
7.
8.  /* 2-2 单行列表项目 */
9.  .list-item{
10.   display: flex;
11.   flex-direction: row;
12.   border-bottom: 1rpx solid gray;
13. }
14.
15. /* 2-3 新闻图片 */
16. .list-item image{
17.   width: 230rpx;
18.   height: 150rpx;
19.   margin: 10rpx;
20. }
21.
22. /* 2-4 新闻标题 */
23. .list-item text{
24.   width: 100%;
25.   line-height: 60rpx;
26.   font-size: 10pt;
27. }
```

16.5.2 公共函数文件代码展示

JS 文件(utils/common.js)的完整代码如下：

```
1.  //服务器域名地址
2.  var baseUrl = 'http:                    //127.0.0.1/myNews/'
3.
4.  //获取新闻列表接口
5.  var getNewsList = baseUrl + 'Index/getNewsList'
6.
7.  //根据新闻 ID 获取新闻内容接口
8.  var getNewsById = baseUrl + 'Index/getNewsById'
9.
10. /**
11.  * 自定义函数 -- 跳转新闻浏览页面
```

```
12.  */
13. function goToDetail(id) {
14.   wx.navigateTo({
15.     url: '../detail/detail?id=' + id,
16.   })
17. }
18.
19. module.exports = {
20.   getNewsList: getNewsList,
21.   getNewsById: getNewsById,
22.   goToDetail: goToDetail
23. }
```

16.5.3 页面文件代码展示

1 首页代码展示

WXML 文件(pages/index/index.wxml)的完整代码如下：

```
1.  <!-- 幻灯片 -->
2.  <swiper indicator-dots autoplay interval='5000' duration='500'>
3.    <block wx:for='{{[newsList[0],newsList[1],newsList[2]]}}' wx:key='swiper{{index}}'>
4.      <swiper-item>
5.        <image src='{{item.poster}}'></image>
6.      </swiper-item>
7.    </block>
8.  </swiper>
9.
10. <!-- 新闻列表 -->
11. <view id='news-list'>
12.   <view class='list-item' wx:for='{{newsList}}' wx:key='news{{index}}' wx:for-item=
'news'>
13.     <image src='{{news.poster}}'></image>
14.     <text bindtap='goToDetail' data-id='{{news.id}}'>
            ◇{{news.title}}——{{news.add_date}}</text>
15.   </view>
16.   <button plain loading="{{loading}}" bindtap="loadMore">
17.     {{loadMoreText}}
18.   </button>
19. </view>
```

WXSS 文件(pages/index/index.wxss)的完整代码如下：

```
1.  /* swiper 区域样式 */
2.  /* 1-1 swiper 组件 */
3.  swiper {
4.    height: 400rpx;
5.  }
6.  /* 1-2 swiper 中的图片 */
7.  swiper image {
8.    width: 100%;
9.    height: 100%;
10. }
```

JS 文件(pages/index/index.js)的完整代码如下：

```
1.  var common = require('../../utils/common.js')
2.  var isEnd = false
3.  var currentPage = 1
4.  Page({
5.    /**
6.     * 页面的初始数据
7.     */
8.    data: {
9.      loading: false,
10.     loadMoreText: '加载更多'
11.   },
12.   /**
13.    * 自定义函数 -- 跳转新闻浏览页面
14.    */
15.   goToDetail: function(e) {
16.     let id = e.currentTarget.dataset.id
17.     common.goToDetail(id)
18.   },
19.   /**
20.    * 自定义函数 -- 加载更多新闻
21.    */
22.   loadMore: function() {
23.     //如果新闻尚未全部加载完毕,并且按钮不处于正在加载状态
24.     if (!isEnd && !this.data.loading) {
25.       //让按钮出现加载动画
26.       this.setData({
27.         loading: true
28.       })
29.       setTimeout(() => {
30.         //加载当前页面新闻数据
31.         this.getNewsListByPage(currentPage)
32.         //停止按钮加载动画
33.         this.setData({
34.           loading: false
35.         })
36.       }, 1000)
37.     }
38.   },
39.   /**
40.    * 自定义函数 -- 获取指定页数的新闻列表
41.    */
42.   getNewsListByPage: function(page) {
43.     var that = this
44.     wx.request({
45.       url: common.getNewsList,
46.       data: {
47.         page: page
48.       },
49.       success: function(res) {
50.         //获取新闻总数
51.         let total = res.data.total
52.         //追加更多新闻数据
53.         let list = that.data.newsList.concat(res.data.list)
54.         //更新新闻数据和新闻总数
55.         that.setData({
```

```
56.             newsList: list,
57.             total: total
58.           })
59.           //如果已经显示全部新闻
60.           if (list.length == total) {
61.             isEnd = true
62.             that.setData({
63.               loadMoreText: '已无更多'
64.             })
65.           } else {
66.             //翻到下一页
67.             currentPage++
68.           }
69.         }
70.       })
71.   },
72.   /**
73.    * 生命周期函数--监听页面加载
74.    */
75.   onLoad: function(options) {
76.         this.getNewsListByPage(1)                //获取第一页新闻数据
77.   },
78. })
```

2 新闻页代码展示

WXML 文件(pages/detail/detail.wxml)的完整代码如下：

```
1.  <view class='container'>
2.    <!-- 新闻标题 -->
3.    <view class='title'>{{article.title}}</view>
4.    <!-- 新闻图片 -->
5.    <view class='poster'>
6.      <image src='{{article.poster}}' mode='widthFix'></image>
7.    </view>
8.    <!-- 新闻正文 -->
9.    <view class='content'>
10.     <text>{{article.content}}</text>
11.   </view>
12.   <!-- 新闻日期 -->
13.   <view class='add_date'>时间：{{article.add_date}}</view>
14.
15.   <button wx:if='{{isAdd}}' plain bindtap='cancelFavorites'>♥已收藏</button>
16.   <button wx:else plain bindtap='addFavorites'>♥点击收藏</button>
17. </view>
```

WXSS 文件(pages/detail/detail.wxss)的完整代码如下：

```
1.  /*整体容器*/
2.  .container{
3.    padding: 15rpx;
4.    text-align: center;
5.  }
6.  /*新闻标题*/
7.  .title{
8.    font-size: 14pt;
9.    line-height: 80rpx;
```

```
10. }
11. /* 新闻图片 */
12. .poster image{
13.   width: 100%;
14. }
15. /* 新闻正文 */
16. .content{
17.   text-align: left;
18.   font-size: 12pt;
19.   line-height: 60rpx;
20. }
21. /* 新闻日期 */
22. .add_date{
23.   font-size: 12pt;
24.   text-align: right;
25.   line-height: 30rpx;
26.   margin-right: 25rpx;
27.   margin-top: 20rpx;
28. }
29. /* "点击收藏"按钮 */
30. button{
31.   width: 250rpx;
32.   height: 100rpx;
33.   margin: 20rpx auto;
34. }
```

JS 文件(pages/detail/detail.js)的完整代码如下：

```
1.  var common = require('../../utils/common.js')
2.  Page({
3.    //添加到收藏夹
4.    addFavorites: function(options) {
5.      let article = this.data.article;              //获取当前新闻
6.      wx.setStorageSync(article.id, article);       //添加到本地缓存
7.      this.setData({isAdd: true});                  //更新按钮显示
8.    },
9.    //取消收藏
10.   cancelFavorites: function() {
11.     let article = this.data.article;              //获取当前新闻
12.     wx.removeStorageSync(article.id);             //从本地缓存删除
13.     this.setData({isAdd: false});                 //更新按钮显示
14.   },
15.   /**
16.    * 生命周期函数--监听页面加载
17.    */
18.   onLoad: function(options) {
19.     //获取携带的新闻 ID 编号
20.     let id = options.id
21.     //检查当前新闻是否在收藏夹中
22.     var article = wx.getStorageSync(id)
23.     //已存在
24.     if (article != '') {
25.       this.setData({
26.         isAdd: true,
27.         article: article
28.       })
```

```
29.         }
30.         //不存在
31.         else {
32.           this.setData({isAdd: false})
33.           var that = this
34.           //向服务器发出请求获取新闻
35.           wx.request({
36.             url: common.getNewsById,
37.             data: {
38.               id: id
39.             },
40.             success: function(res) {
41.               that.setData({ article: res.data[0] })
42.             }
43.           })
44.         }
45.       },
46.   })
```

3 个人中心页代码展示

WXML 文件(pages/my/my.wxml)的完整代码如下：

```
1.  <!-- 登录面板 -->
2.  <view id='myLogin'>
3.    <block wx:if='{{isLogin}}'>
4.      <image id='myIcon' src='{{src}}'></image>
5.      <text id='nickName'>{{nickName}}</text>
6.    </block>
7.    <button wx:else open-type='getUserInfo' bindgetuserinfo='getMyInfo'>
8.      未登录,点此登录
9.    </button>
10. </view>
11.
12. <!-- 我的收藏 -->
13. <view id='myFavorites'>
14.   <text>我的收藏({{num}})</text>
15.   <!-- 新闻列表 -->
16.   <view id='news-list'>
17.     <view class='list-item' wx:for='{{newsList}}' wx:key='news{{index}}' wx:for-item=
'news'>
18.       <image src='{{news.poster}}'></image>
19.       <text bindtap='goToDetail' data-id='{{news.id}}'>
              ◇{{news.title}}——{{news.add_date}}</text>
20.     </view>
21.   </view>
22. </view>
```

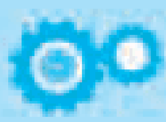
WXSS 文件(pages/my/my.wxss)的完整代码如下：

```
1.  /*登录面板*/
2.  #myLogin{
3.    background-color: #328EEB;
4.    height: 400rpx;
5.    display: flex;
6.    flex-direction: column;
7.    align-items: center;
```

```
8.     justify-content: space-around;
9.   }
10.  /*1-1 头像图片*/
11.  #myIcon{
12.    width: 200rpx;
13.    height: 200rpx;
14.    border-radius: 50%;
15.  }
16.  /*1-2 微信昵称*/
17.  #nickName{
18.    color: white;
19.  }
20.  /*我的收藏*/
21.  #myFavorites{
22.    padding: 20rpx;
23.  }
```

JS 文件(pages/my.my.js)的完整代码如下：

```
1.   var common = require('../../utils/common.js')
2.   Page({
3.     /**
4.      * 页面的初始数据
5.      */
6.     data: {
7.       num: 0,
8.       isLogin: false
9.     },
10.    //获取收藏列表
11.    getMyFavorites: function() {
12.      let info = wx.getStorageInfoSync()             //读取本地缓存信息
13.      let keys = info.keys                           //获取全部 key 信息
14.      let num = keys.length                          //获取收藏新闻数量
15.
16.      let myList = []
17.      for (var i = 0; i < num; i++) {
18.        let obj = wx.getStorageSync(keys[i])
19.        myList.push(obj)                             //将新闻添加到数组中
20.      }
21.      //更新收藏列表
22.      this.setData({
23.        newsList: myList,
24.        num: num
25.      })
26.    },
27.    //获取微信用户信息
28.    getMyInfo: function(e) {
29.      let info = e.detail.userInfo
30.      this.setData({
31.        isLogin: true,                               //确认登录状态
32.        src: info.avatarUrl,                         //更新图片来源
33.        nickName: info.nickName                      //更新昵称
34.      })
35.      //获取收藏列表
36.      this.getMyFavorites()
37.    },
```



```
38.    /**
39.      * 自定义函数 -- 跳转新闻浏览页面
40.      */
41.    goToDetail: function(e) {
42.      let id = e.currentTarget.dataset.id
43.      common.goToDetail(id)
44.    },
45.    /**
46.      * 生命周期函数 -- 监听页面显示
47.      */
48.    onShow: function() {
49.      if (this.data.isLogin) {
50.        //获取收藏列表
51.        this.getMyFavorites();
52.      }
53.    },
54.  })
```

第17章 Chapter 17

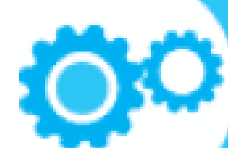


小程序云开发·基于云数据库的高校新闻网

为了让开发者能够更专注于小程序前端的开发工作，腾讯提供了云开发平台帮助开发者快速上线和迭代小程序项目。开发者开通云开发环境后无须自行搭建第三方服务器即可享有存储、数据库等云能力。本章将从零开始详解如何开通小程序云环境，并使用云数据库改造前面第15章的高校新闻网小程序项目。

本章学习目标

- 了解什么是云开发，掌握云开发的开通步骤；
- 掌握创建云模板项目和老项目迁移的步骤；
- 掌握云数据集创建和权限设定的步骤；
- 掌握小程序前端与云数据库交互的方法。



17.1 云开发简介



17.1.1 什么是云开发

云开发是腾讯提供的一套完整原生云端支持和微信服务支持，开发者无须搭建第三方服务器，可以直接使用云端能力开发微信小程序、小游戏。

云开发弱化了后端和运维的概念，开发者可以更多地专注于小程序前端开发，直接使用云平台提供的API进行核心业务开发，并实现快速上线和迭代。该能力与开发者使用的其他云服务可以互相兼容，即可以同时使用云开发能力和其他第三方云服务。

17.1.2 云开发能力介绍

目前云开发能力主要分为三大功能，即云函数、云存储和云数据库，具体解释如下。

- 云函数：在云端运行的函数代码，开发者无须自建服务器，微信私有协议天然鉴权；
- 云存储：小程序前端可以上传/下载云端文件，开发者无须自建存储和CDN，在云开发控制台可视化管理；
- 云数据库：以MongoDB为基础的JSON数据库，开发者无须自建数据库，可以直接在小程序前端或云函数中对数据库进行读写管理。

云开发能力提供的免费资源配额如表 17-1 所示。

表 17-1 云开发免费资源配额

云开发能力的分类	配额种类	额度
存储	容量	5GB
	下载操作次数	150 万次/月
	上传操作次数	60 万次/月
	外网下行流量	无
	CDN 回源流量	5 吉字节/月
CDN	CDN 流量	5 吉字节/月
云函数	调用次数	20 万次/月
	数量	50 个
	资源使用量(GBs)	4 万/月
	外网出流量	1 吉字节/月
	单个云函数并发数	20
数据库	容量	2GB
	QPS	30
	同时连接数	20
	读操作次数	5 万次/天
	写操作次数	3 万次/天
	集合限制	100 个
	单个集合索引限制	20 个

注意：以上是单个云环境的配额，不是全部云环境的总和。

17.1.3 云开发的开通步骤

用户可以按照以下 3 个步骤开通云开发能力。

1 新建云开发模板

首先开发者使用微信 web 开发者工具新建一个空白项目，必须填入 AppID(不可使用游客模式或测试号)，然后选中“小程序 · 云开发”，即可生成一个自带云开发模板样例代码的云开发项目。

云开发项目在自动生成后目录结构和普通小程序不同，根目录中会出现 cloudfunctionRoot 目录用于存放云函数，并且在 project.config.json 文件中新增了该目录的同名字段。

需要注意的是，云能力要求基础库 2.2.3 及以上版本的支持。

2 开通云开发

在创建完第一个云开发项目后，还需要手动开通云开发功能才可继续使用。单击微信 web 开发者工具顶端的“云开发”按钮打开云控制台，根据提示填写开通云开发。在默认情况下可以免费创建两个云环境(创建的第一个环境自动成为默认环境)，每个环境都有唯一的 ID 表示，并且独立包含数据库、存储空间和云函数配置等资源，彼此隔离，互不干扰。

需要注意的是，首次开通云环境后需要等待 10 分钟左右才可正常使用云 API。开通完成后即可在微信 web 开发者工具的模拟器上体验云开发的相关能力。

3 云控制台管理

开通云环境后单击微信 web 开发者工具顶端的“云开发”按钮即可进入云控制台进行可视化管理，如图 17-1 所示。

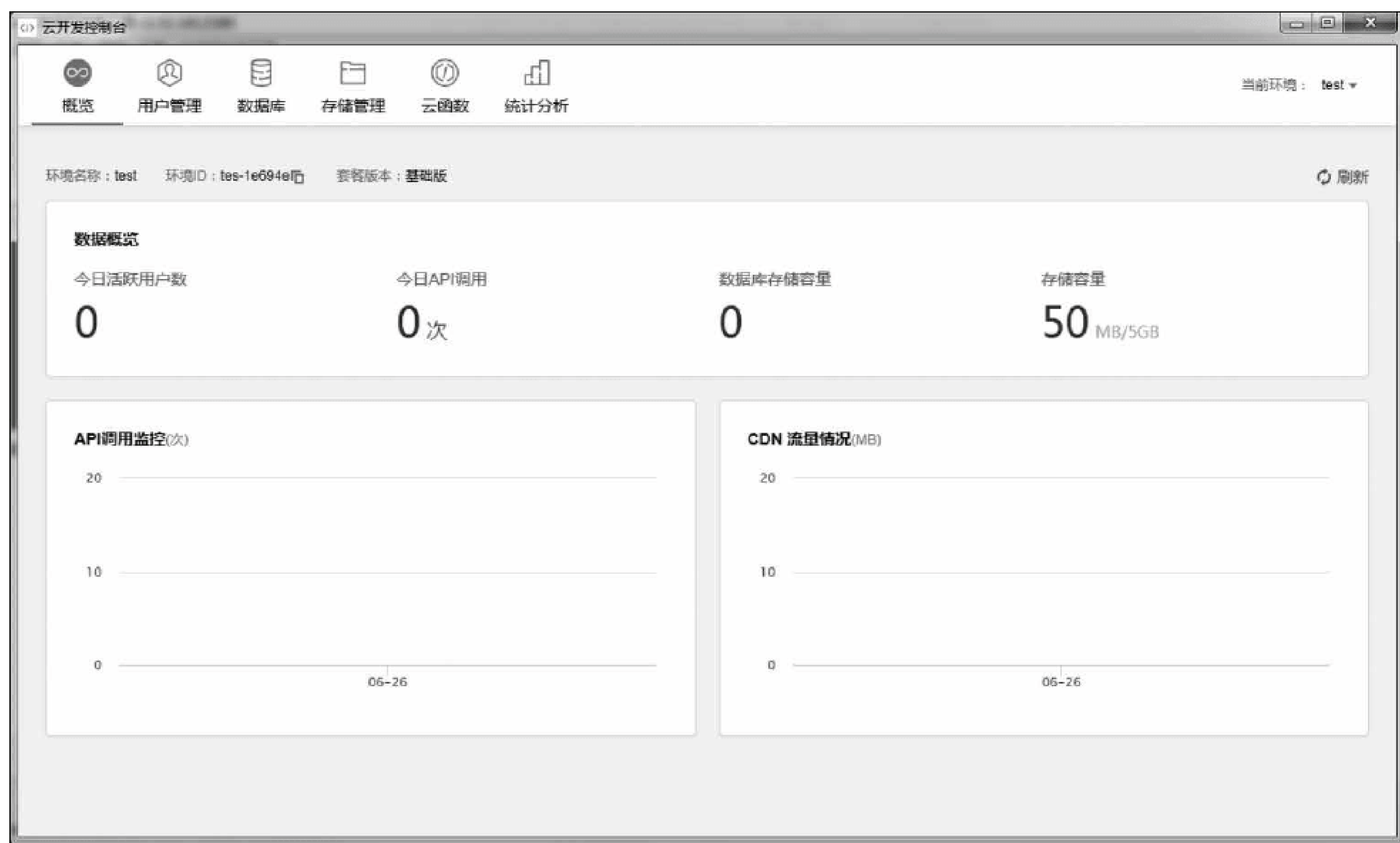


图 17-1 云开发控制台概览图

控制台主要提供以下 6 个选项。

- 概览：查看当前环境名称、ID 和套餐版本，以及数据概览信息，包括今日活跃用户数、API 调用、数据库容量、存储容量和 CDN 流量情况等；
- 用户管理：查看和管理使用小程序项目的用户信息，包括用户的头像、性别、昵称、城市、注册时间、最后一次进入时间以及用户的 openid；
- 数据库：查看和管理数据库，包括对其中数据集的 CRUD 操作和访问权限管理；
- 存储管理：查看和管理存储空间，可以分配访问权限；
- 云函数：查看和管理云函数列表、配置和日志情况等；
- 统计分析：分别展示 API 调用、数据库、存储和云函数的调用情况或资源使用量。



17.2 初始化项目



17.2.1 创建云模板项目



视频讲解

首先需要创建一个云开发项目，在任意盘符下创建一个空白文件夹（例如 cloudNews），然后填入 AppID 和选中“小程序 · 云开发”，如图 17-2 所示。

接着删除其中无用的模板代码：

（1）删除 cloudFunctions 文件夹下的默认云函数 login 的全部内容。

（2）找到 app.json 文件，打开并删除其中的页面引用，只保留第一个 pages/index/index。

（3）打开硬盘中的 pages 文件夹，删除 index 以外的所有目录。

（4）进入 index 文件夹，删除多余的图片，以及 JS、WXML 和 WXSS 文件中的全部代码

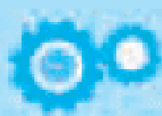




图 17-2　云模板项目填写效果示意图

(JS 文件删除干净后输入关键词 page 让它自动生成 Page({})函数的相关代码)。

(5) 删除 app. wxss 中的代码。

(6) 删除 image 文件夹中的所有图片。

(7) 删除 style 文件夹，此时项目清理全部完成。

17.2.2　迁移老项目



视频讲解

现在需要将第 15 章 newsDemo 中的相关文件合并到当前新建的云项目中，具体步骤如下：

(1) 将 newsDemo 中 images 文件夹内的图片复制、粘贴过来。

(2) 将 newsDemo 中的 utils 文件夹复制、粘贴过来。

(3) 将 newsDemo 中 pages 文件夹内的所有内容复制、粘贴过来，其中 index 文件夹全部替换。

(4) 将 newsDemo 中的 app. json 和 app. wxss 文件替换过来，此时项目迁移完成。

17.2.3　部署云数据库



视频讲解

具体操作如下：

(1) 将若干条新闻数据做到 Excel 表格里面，第一行为标题。

(2) 将 Excel 表格文件另存为 CSV 格式。

(3) 安装 Notepad++软件，打开 CSV 文件，转换为 utf8 编码格式，并保存。

(4) 打开云开发控制台，创建一个新的数据集，例如 news。

(5) 检查 news 数据集的权限，确认权限选择的是“所有用户可读，仅创建者及管理员可写”。

（6）导入 CSV 文件，完成。

效果如图 17-3 所示。

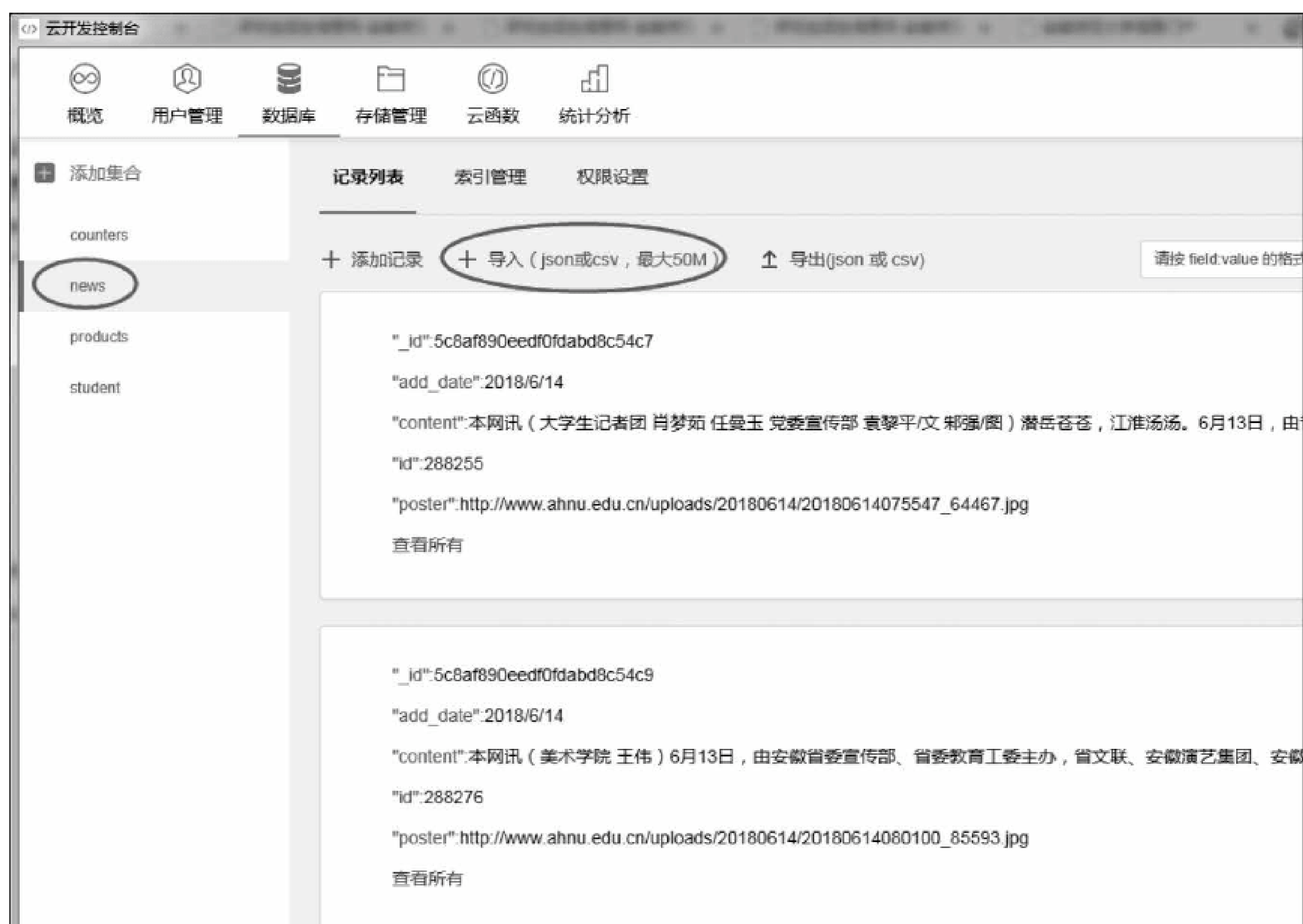


图 17-3 云数据库中的 news 数据集效果图

主要字段参考如下，开发者也可以根据实际需要自行修改或追加其他字段。

- title：新闻标题；
- add_date：添加日期；
- content：正文内容；
- poster：新闻图片。

 17.3 首页改造 

17.3.1 展示新闻列表



视频讲解

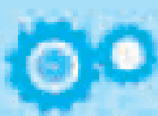
首先需要去掉原来的临时数据，修改 index.js 文件，清空其中的 data 初始数据：

```
1.    /**
2.     * 页面的初始数据
3.     */
4.    data: {
5.      //这里的所有数据清空即可
6.    },
```

然后进行云数据库的新闻读取，修改 index.js 文件，在其顶部添加如下代理：

```
1.  const db = wx.cloud.database()
2.  const news = db.collection('news')
```

修改 index.js 中的 onLoad 函数：

```
1.    /**
2.     * 生命周期函数 -- 监听页面加载
3.     */
4.    onLoad: function(options) {
5.      news.limit(5).get({
6.        success:res =>{
7.          this.setData({newsList:res.data})
8.        }
9.      })
10.   },
```

上述代码表示从云数据库的 news 数据集中读取前 5 条新闻记录，开发者可以修改 limit(5)里面的数字。此时重新运行，新闻列表成功读取到，如图 17-4 所示。

17.3.2 展示滚动图片



视频讲解

滚动图片可以由开发者定义显示最新的 3～5 张，以 3 张为例，修改代码如下：

```
1.  <!-- 幻灯片 -->
2.  <swiper indicator-dots autoplay interval='5000' duration='500'>
3.    <block wx:for='{{[newsList[0],newsList[1],newsList[2]]}}' wx:key='swiper{{index}}'>
4.      <swiper-item>
5.        <image src='{{item.poster}}'></image>
6.      </swiper-item>
7.    </block>
8.  </swiper>
```

此时重新运行，滚动图片成功显示，如图 17-5 所示。



图 17-4 首页新闻列表展示效果图



图 17-5 首页顶端滚动图片效果图

17.3.3 触底自动加载新闻列表



视频讲解

首先修改 index.json 文件，在其中添加新属性 onReachBottomDistance：

```
1.  {
2.    "onReachBottomDistance":100
3.  }
```

上述代码表示一旦页面下拉触到底部 100 像素的位置，就触发触底事件。

修改 index.js 文件，在顶端声明当前页面 page 和每次读取新闻数量 row：

```
1.  const row = 5
2.  var page = 0
```

修改 index.js 中的 onLoad 函数，将固定值 limit(5)改成 limit(row)：

```
1.      /**
2.       * 生命周期函数--监听页面加载
3.       */
4.      onLoad: function(options) {
5.        news.limit(row).get({
6.          success:res=>{
7.            this.setData({newsList:res.data})
8.          }
9.        })
10.     },
```

修改 index.js 中的 onReachBottom 函数，每次更新新闻列表：

```
1.      /**
2.       * 页面下拉触底事件的处理函数
3.       */
4.      onReachBottom: function() {
5.        //翻下一页
6.        page++
7.        //获取当前页面的新闻记录
8.        news.skip(row * page).limit(row).get({
9.          success: res => {
10.           //获取原有的新闻记录
11.           let old_data = this.data.newsList
12.           //获取新页面的新闻记录
13.           let new_data = res.data
14.           //更新首页新闻列表
15.           this.setData({
16.             newsList: old_data.concat(new_data)
17.           })
18.         }
19.       })
20.     },
```

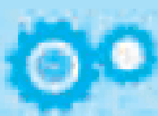
此时拉到页面底部就可以更新出新闻列表了。

17.3.4 点击新闻列表传递新闻编号



视频讲解

如果希望阅读页面显示不同的新闻内容，需要知道新闻编号。

修改 index. wxml 如下：

```
1.  <!-- 新闻列表 -->
2.  <view id = 'news - list'>
3.      <view class = 'list - item' wx:for = '{{newsList}}' wx:key = 'news{{index}}' wx:for - item = 
'news'>
4.          <image src = '{{news.poster}}'></image>
5.          <text data - id = '{{news._id}}' bindtap = 'goToDetail'>
                ◇{{news.title}} -- {{news.add_date}}</text>
6.      </view>
7.  </view>
```

上述代码表示点击新闻标题时附带 ID 编号，然后修改 index. js 中的 goToDetail 函数：

```
1.    /**
2.     * 自定义函数 -- 跳转新闻浏览页面
3.     */
4.    goToDetail: function(e) {
5.      common.goToDetail(e.currentTarget.dataset.id)
6.    },
```

然后更新 utils 文件夹中的 common. js 文件，删除已经不需要的模拟数据 const news，以及函数 getNewsList 和 getNewsDetail，并新增自定义函数 goToDetail，用于跳转新闻浏览页面：

```
1.  /**
2.   * 自定义函数 -- 跳转新闻浏览页面
3.   */
4.  function goToDetail(id) {
5.    wx.navigateTo({
6.      url: '../detail/detail?id = ' + id,
7.    })
8.  }
9.
10. module.exports = {
11.   goToDetail: goToDetail
12. }
```

这样点击跳转到 detail 页面时就可以传递新闻编号。



17.4 新闻阅读页改造



新闻阅读页的改造主要集中在如何获取新闻正文内容。



视频讲解

修改 detail. js 文件，在其顶部添加如下代码：

```
1.  const db = wx.cloud.database()
2.  const news = db.collection('news')
```

然后修改 detail. js 中的 onLoad 函数：

```
1.    /**
2.     * 生命周期函数 -- 监听页面加载
3.     */
4.    onLoad: function(options) {
```

```
5.        //显示 loading 提示框
6.        wx.showLoading({
7.          title: '数据加载中'
8.        })
9.
10.       //获取新闻编号
11.       let id = options.id
12.       //根据新闻 ID 查找新闻是否在收藏夹中
13.       let article = wx.getStorageSync(id)
14.
15.       //新闻在收藏夹中
16.       if (article != '') {
17.         //更新页面上的新闻信息和收藏状态
18.         this.setData({
19.           article: article,
20.           isAdd: true
21.         })
22.         //隐藏 loading 提示框
23.         wx.hideLoading()
24.       }
25.       //新闻不在收藏夹中
26.       else {
27.         //根据新闻 ID 在云数据集中查找新闻内容
28.         news.doc(id).get({
29.           success: res => {
30.             //更新页面上的新闻信息和收藏状态
31.             this.setData({
32.               article: res.data,
33.               isAdd: false
34.             })
35.             //隐藏 loading 提示框
36.             wx.hideLoading()
37.           }
38.         })
39.       }
40.     },
```

此时重新运行，新闻内容成功读取到，如图 17-6 所示。



图 17-6　新闻阅读页成功读取到新闻信息

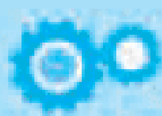



个人中心页的改造主要集中在如何浏览已收藏的新闻，与首页的点击跳转新闻内容功能类似，首先修改 my.wxml 中收藏列表的代码：



视频讲解

```
1.    <!-- 新闻列表 -->
2.    <view id = 'news - list'>
3.      <view class = 'list - item' wx:for = '{{newsList}}' wx:key = 'news
{{index}}' wx:for - item = 'news'>
4.        <image src = '{{news.poster}}'></image>
5.        <text data - id = '{{news._id}}' bindtap = 'goToDetail'>
◇{{news.title}} -- {{news.add_date}}</text>
6.      </view>
```



```
7.      </view>
```

上述代码为<text>组件的 data-id 属性更新了新闻 ID 编号。

然后在对应的 my.js 文件中修改 goToDetail 函数的内容，代码片段如下：

```
1.      /**
2.       * 自定义函数 -- 跳转新闻浏览页面
3.       */
4.      goToDetail: function(e) {
5.        common.goToDetail(e.currentTarget.dataset.id)
6.      },
```

最终运行效果如图 17-7 所示。



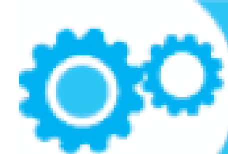
(a) "我的收藏" 列表



(b) 浏览收藏的新闻

图 17-7 浏览已收藏新闻的效果

此时整个项目就全部改造完毕，由于图片地址有可能在示例高校运维网站群平台时发生变化，开发者可以替换成自己所在学校的其他图片的 URL 地址。



17.6 完整代码展示



17.6.1 应用文件代码展示

app.json 文件的完整代码如下：

```
1.  {
2.    "pages": [
3.      "pages/index/index",
4.      "pages/my/my",
```

```
5.        "pages/detail/detail"
6.      ],
7.      "window": {
8.        "navigationBarBackgroundColor": "#328EEB",
9.        "navigationBarTitleText": "我的新闻网"
10.     },
11.     "tabBar": {
12.       "color": "#000",
13.       "selectedColor": "#328EEB",
14.       "list": [
15.         {
16.           "pagePath": "pages/index/index",
17.           "text": "首页",
18.           "iconPath": "images/index.png",
19.           "selectedIconPath": "images/index_blue.png"
20.         },
21.         {
22.           "pagePath": "pages/my/my",
23.           "text": "我的",
24.           "iconPath": "images/my.png",
25.           "selectedIconPath": "images/my_blue.png"
26.         }
27.       ]
28.     }
29. }
```

app.wxss 文件的完整代码如下：

```
1.  /*新闻列表区域样式*/
2.  /*2-1 新闻列表容器*/
3.  #news-list {
4.    min-height: 600rpx;
5.    padding: 15rpx;
6.  }
7.  /*2-2 列表项目*/
8.  .list-item{
9.    display: flex;
10.   flex-direction: row;
11.   border-bottom: 1rpx solid gray;
12. }
13. /*2-3 新闻图片*/
14. .list-item image{
15.   width:230rpx;
16.   height: 150rpx;
17.   margin: 10rpx;
18. }
19. /*2-4 新闻标题*/
20. .list-item text{
21.   width: 100%;
22.   line-height: 60rpx;
23.   font-size: 10pt;
24. }
```

17.6.2 公共函数文件代码展示

JS 文件(utils/common.js)的完整代码如下：

```
1.  /**
2.   * 自定义函数 -- 跳转新闻浏览页面
3.   */
4.  function goToDetail(id) {
5.    wx.navigateTo({
6.      url: '../detail/detail?id=' + id,
7.    })
8.  }
9.
10.
11. module.exports = {
12.   goToDetail: goToDetail
13. }
```

17.6.3 页面文件代码展示

1 首页代码展示

WXML 文件(pages/index/index.wxml)的完整代码如下：

```
1.  <!-- 幻灯片 -->
2.  <swiper indicator-dots autoplay interval='5000' duration='500'>
3.    <block wx:for='{{[newsList[0],newsList[1],newsList[2]]}}' wx:key='swiper{{index}}'>
4.      <swiper-item>
5.        <image src='{{item.poster}}'></image>
6.      </swiper-item>
7.    </block>
8.  </swiper>
9.
10. <!-- 新闻列表 -->
11. <view id='news-list'>
12.   <view class='list-item' wx:for='{{newsList}}' wx:key='news{{index}}' wx:for-item=
'news'>
13.     <image src='{{news.poster}}'></image>
14.     <text bindtap='goToDetail' data-id='{{news._id}}'>
            ◇{{news.title}} -- {{news.add_date}}</text>
15.   </view>
16.   <button plain loading="{{loading}}" bindtap="loadMore">
17.     {{loadMoreText}}
18.   </button>
19. </view>
```

WXSS 文件(pages/index/index.wxss)的完整代码如下：

```
1.  /* swiper 区域样式 */
2.  /* 1-1 swiper 组件 */
3.  swiper {
4.    height: 400rpx;
5.  }
6.  /* 1-2 swiper 中的图片 */
7.  swiper image {
8.    width: 100%;
9.    height: 100%;
10. }
```

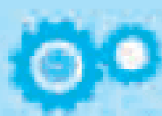
JS 文件(pages/index/index.js)的完整代码如下：

```
1.  var common = require('../../utils/common.js')
2.  const db = wx.cloud.database()
3.  const news = db.collection('news')
4.
5.  var row = 5                    //每次读取的新闻数量
6.  var page = 0                   //当前是第几页
7.
8.  Page({
9.    /**
10.    * 页面的初始数据
11.    */
12.   data: {
13.
14.   },
15.   /**
16.    * 自定义函数 -- 跳转新闻浏览页面
17.    */
18.   goToDetail: function(e) {
19.     common.goToDetail(e.currentTarget.dataset.id)
20.   },
21.   /**
22.    * 生命周期函数 -- 监听页面加载
23.    */
24.   onLoad: function(options) {
25.     news.limit(row).get({
26.       success: res => {
27.         this.setData({
28.           newsList: res.data
29.         })
30.       }
31.     })
32.   },
33. })
```

2 新闻页代码展示

WXML 文件(pages/detail/detail.wxml)的完整代码如下：

```
1.  <view class='container'>
2.    <!-- 新闻标题 -->
3.    <view class='title'>{{article.title}}</view>
4.    <!-- 新闻图片 -->
5.    <view class='poster'>
6.       <image src='{{article.poster}}' mode='widthFix'></image>
7.    </view>
8.    <!-- 新闻正文 -->
9.    <view class='content'>
10.     <text>{{article.content}}</text>
11.   </view>
12.   <!-- 新闻日期 -->
13.   <view class='add_date'>时间：{{article.add_date}}</view>
14.
15.   <button wx:if='{{isAdd}}' plain bindtap='cancelFavorites'>已收藏</button>
16.   <button wx:else plain bindtap='addFavorites'>点击收藏</button>
17. </view>
```



WXSS 文件(pages/detail/detail.wxss)的完整代码如下：

```
1.  /* 整体容器 */
2.  .container{
3.    padding: 15rpx;
4.    text-align: center;
5.  }
6.  /* 新闻标题 */
7.  .title{
8.    font-size: 14pt;
9.    line-height: 80rpx;
10. }
11. /* 新闻图片 */
12. .poster image{
13.   width: 100%;
14. }
15. /* 新闻正文 */
16. .content{
17.   text-align: left;
18.   font-size: 12pt;
19.   line-height: 60rpx;
20. }
21. /* 新闻日期 */
22. .add_date{
23.   font-size: 12pt;
24.   text-align: right;
25.   line-height: 30rpx;
26.   margin-right: 25rpx;
27.   margin-top: 20rpx;
28. }
29. /*“点击收藏”按钮 */
30. button{
31.   width: 250rpx;
32.   height: 100rpx;
33.   margin: 20rpx auto;
34. }
```

JS 文件(pages/detail/detail.js)的完整代码如下：

```
1.  var common = require('../../utils/common.js')
2.  Page({
3.    /**
4.     * 自定义函数 -- 添加到收藏夹
5.     */
6.    addFavorites: function() {
7.      let article = this.data.article
8.      wx.setStorageSync(article._id, article)
9.      this.setData({
10.       isAdd: true
11.     })
12.   },
13.
14.   /**
15.    * 自定义函数 -- 取消收藏
16.    */
17.   cancelFavorites: function() {
```

```
18.     let article = this.data.article
19.     wx.removeStorageSync(article._id)
20.     this.setData({
21.       isAdd: false
22.     })
23.   },
24.
25.   /**
26.    * 生命周期函数--监听页面加载
27.    */
28.   onLoad: function(options) {
29.     //显示 loading 提示框
30.     wx.showLoading({
31.       title: '数据加载中'
32.     })
33.
34.     //获取新闻编号
35.     let id = options.id
36.     //根据新闻 ID 查找新闻是否在收藏夹中
37.     let article = wx.getStorageSync(id)
38.
39.     //新闻在收藏夹中
40.     if (article != '') {
41.       //更新页面上的新闻信息和收藏状态
42.       this.setData({
43.         article: article,
44.         isAdd: true
45.       })
46.       //隐藏 loading 提示框
47.       wx.hideLoading()
48.     }
49.     //新闻不在收藏夹中
50.     else {
51.       //根据新闻 ID 在云数据集中查找新闻内容
52.       news.doc(id).get({
53.         success: res => {
54.           //更新页面上的新闻信息和收藏状态
55.           this.setData({
56.             article: res.data,
57.             isAdd: false
58.           })
59.           //隐藏 loading 提示框
60.           wx.hideLoading()
61.         }
62.       })
63.     }
64.   },
65. })
```

3 个人中心页代码展示

WXML 文件(pages/my/my.wxml)的完整代码如下:

```
1.  <!-- 登录面板 -->
2.  <view id='myLogin'>
3.    <block wx:if='{{isLogin}}'>
```

```
4.        < image id = 'myIcon' src = '{{src}}'></image >
5.        < text id = 'nickName'>{{nickName}}</text >
6.      </block >
7.      < button wx:else open - type = 'getUserInfo' bindgetuserinfo = 'getMyInfo'>
8.        未登录,点此登录
9.      </button >
10.  </view >
11.
12.  <!-- 我的收藏 -->
13.  < view id = 'myFavorites'>
14.     < text >我的收藏({{num}})</text >
15.     <!-- 新闻列表 -->
16.     < view id = 'news - list'>
17.       < view class = 'list - item' wx:for = '{{newsList}}' wx:key = 'news{{index}}' wx:for - item =
'news'>
18.          < image src = '{{news.poster}}'></image >
19.          < text bindtap = 'goToDetail' data - id = '{{news._id}}'>
                  ◇{{news.title}} -- {{news.add_date}}</text >
20.       </view >
21.     </view >
22.  </view >
```

WXSS 文件(pages/my/my.wxss)的完整代码如下：

```
1.   /* 登录面板 */
2.   #myLogin{
3.     background - color: #328EEB;
4.     height: 400rpx;
5.     display: flex;
6.     flex - direction: column;
7.     align - items: center;
8.     justify - content: space - around;
9.   }
10.  /* 1 - 1 头像图片 */
11.  #myIcon{
12.    width: 200rpx;
13.    height: 200rpx;
14.    border - radius: 50%;
15.  }
16.  /* 1 - 2 微信昵称 */
17.  #nickName{
18.    color: white;
19.  }
20.  /* 我的收藏 */
21.  #myFavorites{
22.    padding: 20rpx;
23.  }
```

JS 文件(pages/my/my.js)的完整代码如下：

```
1.   var common = require('../../utils/common.js')
2.   Page({
3.     /**
4.      * 页面的初始数据
5.      */
6.     data: {
```

```
7.     num: 0
8.   },
9.   /**
10.    * 自定义函数 -- 跳转新闻浏览页面
11.    */
12.  goToDetail: function(e) {
13.    common.goToDetail(e.currentTarget.dataset.id)
14.  },
15.  /**
16.    * 自定义函数 -- 获取用户微信个人信息
17.    */
18.  getUserInfo: function(e) {
19.    console.log(e.detail.userInfo)
20.    let info = e.detail.userInfo
21.    this.setData({
22.      isLogin: true,
23.      src: info.avatarUrl,
24.      nickName: info.nickName
25.    })
26.
27.    this.getMyFavorites()
28.  },
29.  /**
30.    * 自定义函数 -- 获取收藏列表
31.    */
32.  getMyFavorites: function() {
33.    let info = wx.getStorageInfoSync()
34.    let keys = info.keys
35.    let num = keys.length      //收藏的新闻总数
36.
37.    let myList = []
38.    for (var i = 0; i < num; i++) {
39.      let obj = wx.getStorageSync(keys[i])
40.      myList.push(obj)
41.    }
42.
43.    this.setData({
44.      num: num,
45.      newsList: myList
46.    })
47.  },
48.  /**
49.    * 生命周期函数 -- 监听页面显示
50.    */
51.  onShow: function() {
52.    //如果已登录,则更新收藏列表
53.    if (this.data.isLogin) {
54.      this.getMyFavorites()
55.    }
56.  },
57. })
```

第18章 Chapter 18

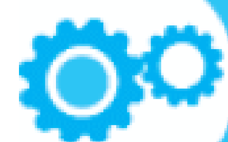


小程序云开发·基于云存储的电子书橱

在了解了云开发的相关能力后，读者不妨尝试将前面第7章的电子书橱小程序改造成基于云存储的小程序项目。改造后的小程序无须自行搭建服务器，可以在小程序前端直接使用云开发API下载云存储中的电子书资源到本地设备进行使用。

本章学习目标

- 掌握云存储和云数据库的管理操作；
- 掌握小程序前端与云数据库交互的方法；
- 掌握小程序前端下载云文件的方法。



18.1 初始化项目



18.1.1 创建云模板项目



视频讲解

首先需要创建一个云开发项目，在任意盘符下创建一个空白文件夹（例如cloudBooks），然后填入AppID和选中“小程序·云开发”，如图18-1所示。

接着删除其中无用的模板代码：

（1）删除cloudFunctions文件夹下的默认云函数login的全部内容。

（2）找到app.json文件，打开并删除其中的页面引用，只保留第一个pages/index/index。

（3）打开硬盘中的pages文件夹，删除index以外的所有目录。

（4）进入index文件夹，删除多余的图片。

（5）删除image文件夹。

（6）删除style文件夹，此时项目清理全部完成。

18.1.2 迁移老项目



视频讲解

现在需要将第7章booksDemo中的相关文件合并到当前新建的云项目中，具体步骤如下：

（1）将booksDemo中pages文件夹内的所有内容复制、粘贴过来，其中index文件夹全部替换。



图 18-1 云模板项目填写效果示意图

(2) 将 booksDemo 中的 app. json 和 app. wxss 文件替换过来，此时项目迁移完成。



视频讲解

18.1.3 部署云文件存储

打开云开发控制台，选择“存储管理”面板，新建文件夹 books，然后进入并将需要的电子书资源(PDF 格式等)上传到云文件存储库中，效果如图 18-2 所示。

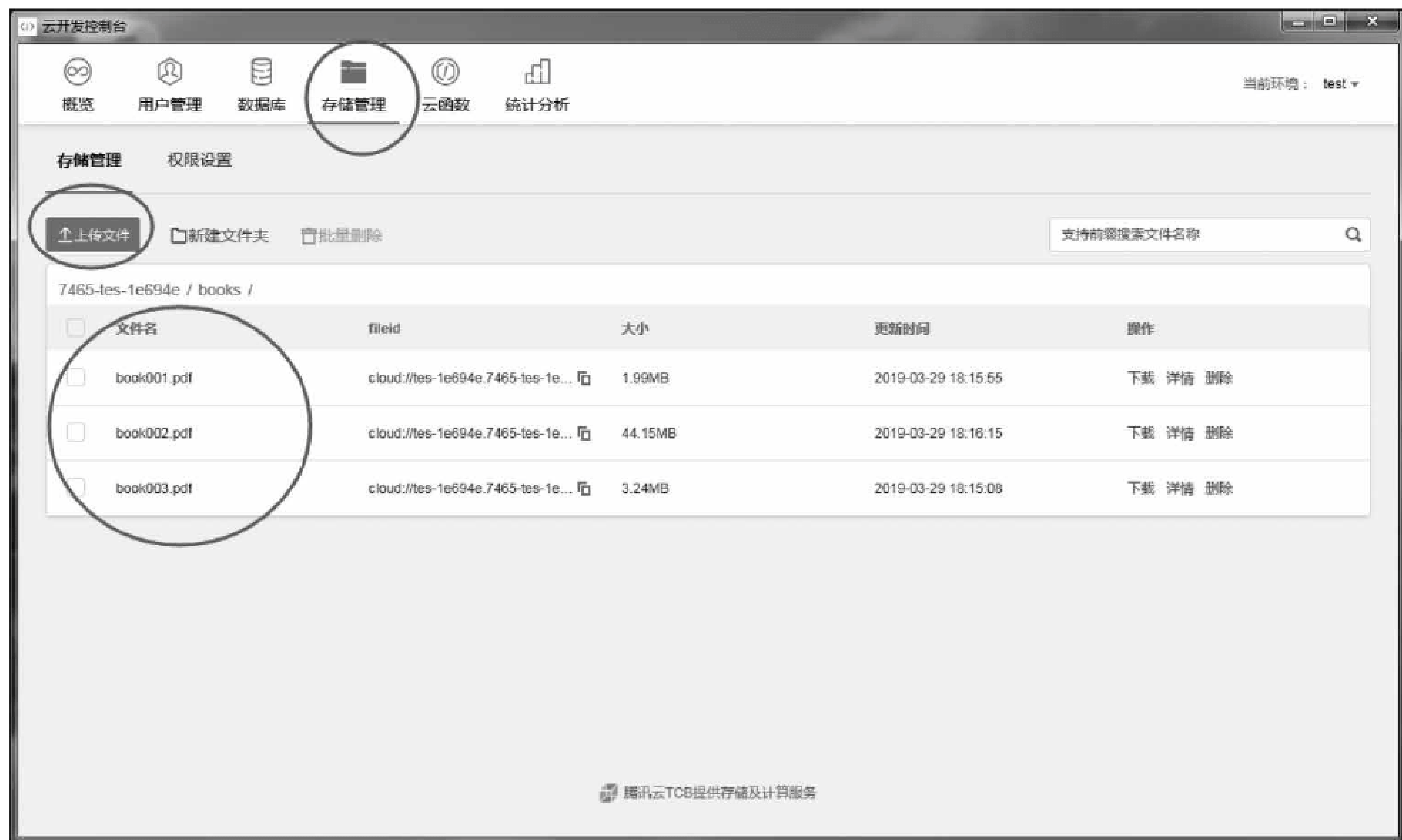
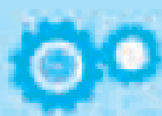


图 18-2 往云存储库中上传电子书示意图

注意：云文件存储库最大允许免费存储 5GB 容量的文件。

18.1.4 部署云数据库


视频讲解

具体操作如下：

（1）将图书数据输入 Excel 表格里面，第一行为标题。

（2）将 Excel 表格文件另存为 CSV 格式。

（3）安装 Notepad++软件，打开 CSV 文件，转换为 utf8 编码格式并保存。

（4）打开云开发控制台，创建一个新的数据集，例如 books。

（5）检查 books 数据集的权限，确认权限选择的是"所有用户可读，仅创建者及管理员可写"。

（6）导入 CSV 文件，完成。

效果如图 18-3 所示。

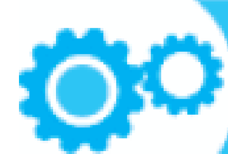


图 18-3 往云数据库中录入图书数据

参考字段如下，对于其他字段，开发者可以根据实际情况自由发挥。

- title：书名；
- author：作者；
- price：价格；
- isbn：书籍的 ISBN 编号；
- coverurl：封面图片地址；
- fileid：云文件存储库中的文件 ID。

其中图书封面地址来源于豆瓣图书频道，开发者也可以自行选择其他素材资源。

 18.2 首页改造 


视频讲解

18.2.1 展示图书列表

首先需要删除原来的临时数据，修改 index.js 文件，清空其中的 data 初

始数据：

```
1.    /**
2.     * 页面的初始数据
3.     */
4.    data: {
5.      bookList: [ ]
6.    },
```

修改 index.js 文件，在其顶部添加如下代码：

```
1.  const db = wx.cloud.database()
2.  const books = db.collection('books')
```

然后修改 index.js 文件中的 onLoad 函数：

```
1.    /**
2.     * 生命周期函数--监听页面加载
3.     */
4.    onLoad: function(options) {
5.      books.get({
6.        success:res =>{
7.          this.setData({bookList:res.data})
8.        }
9.      })
10.   },
```

上述代码表示从云数据集 books 中读取图书信息。

接着修改 index.wxml 页面的代码，为 wx:for 循环内部的元素匹配数据集里的字段：

```
1.  <!-- 图书展示容器 -->
2.  <view class='book-container'>
3.    <!-- 图书单元区域 -->
4.    <view class='box' wx:for='{{bookList}}' wx:key=
'book{{index}}' data-id = '{{item._id}}' bindtap =
'readBook'>
5.      <!-- 图书封面 -->
6.      <image src='{{item.coverurl}}'></image>
7.      <!-- 图书标题文本 -->
8.      <text>{{item.title}}</text>
9.    </view>
10. </view>
```

此时重新运行，图书列表成功读取到，如图 18-4 所示。



图 18-4 图书封面展示效果图

18.2.2 点击跳转图书详情页

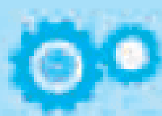


视频讲解

创建页面 intro，用于显示图书详情。

然后在 index.wxml 页面修改点击事件：

```
1.  <!-- 图书展示容器 -->
2.  <view class='book-container'>
3.    <!-- 图书单元区域 -->
4.    <view class='box' wx:for='{{bookList}}' wx:key=
```



```
'book{{index}}' data-id='{{item._id}}' bindtap='showBookIntro'>
5.         <!-- 图书封面 -->
6.         <image src='{{item.coverurl}}'></image>
7.         <!-- 图书标题文本 -->
8.         <text>{{item.title}}</text>
9.       </view>
10.  </view>
```

接下来在 index.js 文件中添加自定义函数 showBookIntro：

```
1.      /**
2.        * 自定义函数--显示图书详情
3.        */
4.      showBookIntro: function(e) {
5.        //获取图书 ID 编号
6.        let id = e.currentTarget.dataset.id
7.        wx.navigateTo({
8.          url: '../intro/intro?id=' + id,
9.        })
10.     },
```

此时点击可以成功跳转图书详情页面，并携带当前图书的 ID 编号。



18.3 图书详情页改造



18.3.1 页面设计



视频讲解

首先将 index.wxml 里面的蒙层代码剪切、粘贴过来使用：

```
1.  <!-- 下载时的蒙层 -->
2.  <view class='loading-container' wx:if='{{isDownloading}}'>
3.    <text>下载中，请稍候</text>
4.     <progress percent="{{percentNum}}" stroke-width="6" activeColor="#663366"
backgroundColor="#FFFFFF" show-info active active-mode="forwards"></progress>
5.  </view>
```

将 index.js 文件中 data 里面的初始数据剪切、粘贴到当前 intro.js 文件的 data 中：

```
1.      /**
2.        * 页面的初始数据
3.        */
4.      data: {
5.        isDownloading: false,
6.        percentNum: 0
7.      },
```

同样将 index.wxss 文件中关于蒙层的样式代码剪切、粘贴到当前 intro.wxss 文件中：

```
1.  /* 下载时的蒙层容器样式 */
2.  .loading-container {
3.    height: 100vh;                  /* 高度 */
4.    background-color: silver;       /* 背景颜色 */
5.    display: flex;                  /* flex 模型布局 */
6.    flex-direction: column;         /* 水平排列 */
```

```
7.    align-items: center;            /* 水平方向居中 */
8.    justify-content: space-around;  /* 分散布局 */
9.  }
10. /* 进度条样式 */
11. progress{
12.   width: 80%;                     /* 宽度 */
13. }
```

然后在 intro.wxml 中追加当前页面的图书详情显示：

```
1.  <!-- 下载时的蒙层(代码略) -->
2.
3.  <!-- 图书详情 -->
4.  <view class='intro-container' wx:else>
5.    <!-- 图书封面图片 -->
6.    <image src='{{book.coverurl}}' mode='widthFix'></image>
7.    <!-- 图书信息介绍 -->
8.    <view class='intro-box'>
9.      <text>书名: {{book.title}}</text>
10.     <text>作者: {{book.author}}</text>
11.     <text>价格: {{book.price}}</text>
12.     <text>ISBN: {{book.isbn}}</text>
13.   </view>
14.   <!-- "开始阅读"按钮 -->
15.   <button type='warn' bindtap='readBook'>开始阅读</button>
16. </view>
```

对应的 intro.wxss 代码如下：

```
1.  /* 图书详细信息区域 */
2.  .intro-container{
3.    height: 100vh;
4.    display: flex;
5.    flex-direction: column;
6.    align-items: center;
7.    justify-content: space-evenly;
8.  }
9.  /* 图书封面图片 */
10. .intro-container image{
11.   width: 400rpx;
12.   margin: 20rpx;
13. }
14. /* 图书信息区域 */
15. .intro-box{
16.   display: flex;
17.   flex-direction: column;
18. }
19. /* 图书文字信息 */
20. .intro-box text{
21.   margin: 20rpx;
22. }
```

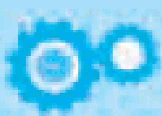
此时页面设计就全部完成。

18.3.2　页面逻辑



视频讲解

修改 intro.js 文件，在其顶部添加如下代码：



```
1.  const db = wx.cloud.database()
2.  const books = db.collection('books')
```

然后修改 intro.js 中的 onLoad 函数：

```
1.    /**
2.     * 生命周期函数--监听页面加载
3.     */
4.    onLoad: function(options) {
5.      books.doc(options.id).get({
6.        success:res=>{
7.          this.setData({book:res.data})
8.        }
9.      })
10.   },
```



图 18-5　查询图书详情页

上述代码表示根据页面跳转过来时所携带的图书 ID 查找云数据集 books 中该图书的相关信息。此时重新运行，图书内容成功读取到，如图 18-5 所示。

由该图可见查询成功，图书信息会根据开发者上传的数据的不同显示实际值。

18.3.3　阅读图书功能



视频讲解

修改 intro.js 代码，将原先在 index.js 中的几个函数剪切、粘贴过来，函数的具体内容略。

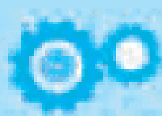
```
1.  /**
2.   * 打开图书
3.   */
4.    openBook: function(path) {代码略},
5.
6.    /**
7.     * 保存图书
8.     */
9.    saveBook: function(id, path) {代码略},
10.
11.   /**
12.    * 阅读图书
13.    */
14.   readBook: function(e) {代码略},
15.
16.   /**
17.    * 封装 showModal 方法
18.    */
19.   showTips: function(content) {代码略},
```

然后修改其中的 readBook 函数：

```
1.    /**
2.     * 阅读图书
3.     */
4.    readBook: function(e) {
5.      var that = this
```

```
6.        //获取当前点击的图书的 ID
7.        let id = this.data.book._id
8.        //查看本地缓存
9.        let path = wx.getStorageSync(id)
10.       //未曾下载过
11.       if (path == '') {
12.         //切换到下载时的蒙层
13.         this.setData({
14.           isDownloading: true
15.         })
16.
17.         //获取当前点击的图书的 URL 地址
18.         let fileid = this.data.book.fileid
19.
20.         const downloadTask = wx.cloud.downloadFile({
21.           fileID:fileid,
22.           success:res =>{
23.             //下载成功
24.             if (res.statusCode == 200) {
25.               //获取地址
26.               path = res.tempFilePath
27.               //保存并打开图书
28.               this.saveBook(id, path)
29.             }
30.             //连上了服务器,下载失败
31.             else {
32.               this.showTips('暂时无法下载!')
33.             }
34.           },fail:err =>{
35.             this.showTips('无法连接到服务器!')
36.           },complete:res =>{
37.             //关闭下载时的蒙层
38.             this.setData({
39.               isDownloading: false
40.             })
41.           }
42.         })
43.
44.         //监听当前文件的下载进度
45.         downloadTask.onProgressUpdate(function(res) {
46.           let progress = res.progress;
47.           that.setData({
48.             percentNum: progress
49.           })
50.         })
51.       }
52.       //之前下载过的,直接打开
53.       else {
54.         //打开图书
55.         this.openBook(path)
56.       }
57.     },
```

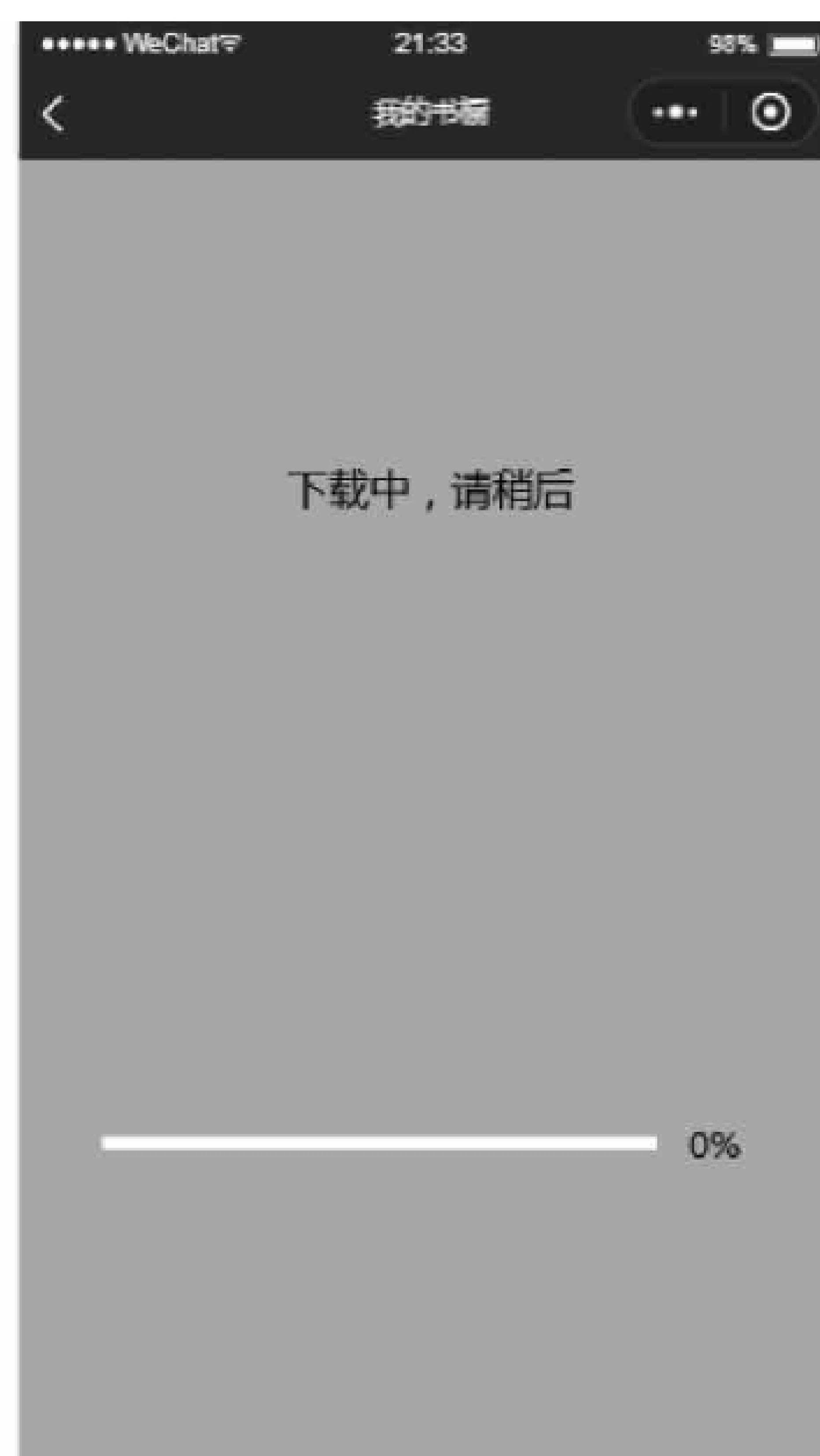
此时点击按钮会检测当前这本图书是否已经下载过了，如果已经下载过了，则直接打开阅读；如果没有下载过，则显示下载进度条蒙层进行下载。



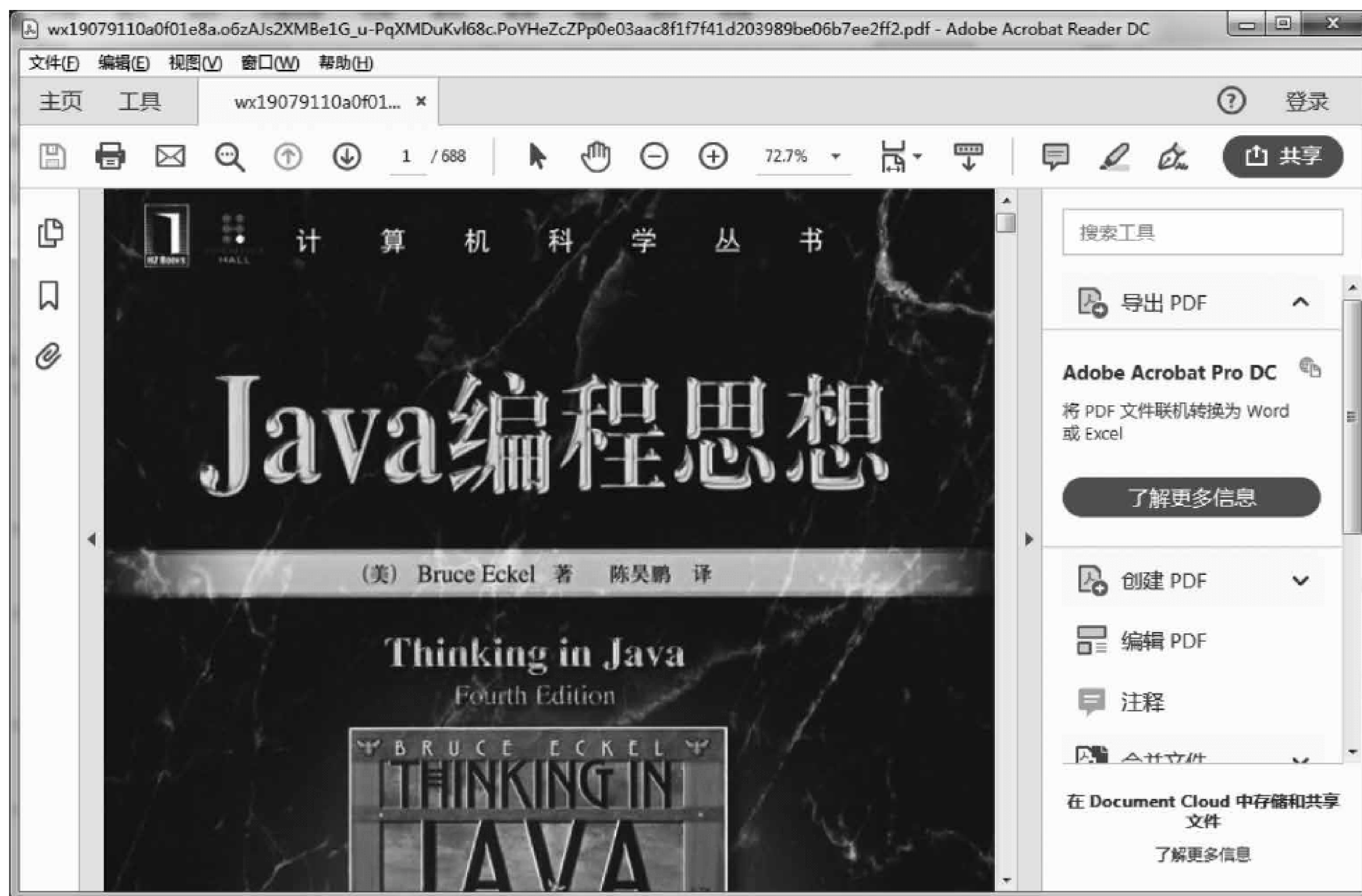
运行效果如图 18-6 所示。



(a) 图书详情页



(b) 正在下载电子书



(c) 在计算机端打开已下载的电子书效果

图 18-6 图书下载过程示意图

由图可见，所选择的电子书已经从云存储空间中下载到本地。用户要注意小程序只允许下载、保存 10MB 以内大小的文件，必要时可以在微信 web 开发者工具中清除文件缓存。

此时基于云存储的电子书橱项目就全部改造完毕了。

 18.4 完整代码展示 

18.4.1 应用文件代码展示

app.json 文件的完整代码如下：

```
1.  {
2.    "pages": [
3.      "pages/index/index",
4.      "pages/intro/intro"
5.    ],
6.    "window": {
7.      "navigationBarBackgroundColor": "#663366",
8.      "navigationBarTitleText": "我的书橱"
9.    }
10. }
```

18.4.2 页面文件代码展示

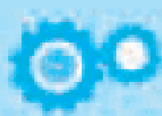
1 首页代码展示

WXML 文件(pages/index/index.wxml)的完整代码如下：

```
1.  <!-- 图书展示容器 -->
2.  <view class='book-container'>
3.    <!-- 图书单元区域 -->
4.    <view class='box' wx:for='{{bookList}}' wx:key='book{{index}}' data-id='{{item._id}}'
bindtap='showBookIntro'>
5.      <!-- 图书封面 -->
6.      <image src='{{item.coverurl}}'></image>
7.      <!-- 图书标题文本 -->
8.      <text>{{item.title}}</text>
9.    </view>
10. </view>
```

WXSS 文件(pages/index/index.wxss)的完整代码如下：

```
1.  /* 图书展示容器样式 */
2.  .book-container {
3.    display: flex;                          /* flex 模型布局 */
4.    flex-direction: row;                    /* 水平排列 */
5.    flex-wrap: wrap;                        /* 允许换行 */
6.  }
7.  /* 图书单元区域样式 */
8.  .box {
9.    width: 50%;                             /* 宽度 */
10.   height: 400rpx;                         /* 高度 */
11.   display: flex;                          /* flex 模型布局 */
12.   flex-direction: column;                 /* 垂直排列 */
13.   align-items: center;                    /* 水平方向居中 */
14.   justify-content: space-around;          /* 分散布局 */
15. }
16. /* 图书封面图片样式 */
```

```
17. image {
18.   width: 200rpx;                    /* 宽度 */
19.   height: 300rpx;                   /* 高度 */
20. }
21. /* 图书标题文本样式 */
22. text {
23.   text-align: center;               /* 文本居中显示 */
24. }
```

JS 文件(pages/index/index.js)的完整代码如下：

```
1.  const db = wx.cloud.database()
2.  const books = db.collection('books')
3.
4.  Page({
5.    /**
6.     * 页面的初始数据
7.     */
8.    data: {
9.      bookList: []
10.   },
11.   /**
12.    * 自定义函数--跳转图书详情页面
13.    */
14.   showBookIntro: function(e) {
15.     let id = e.currentTarget.dataset.id
16.     wx.navigateTo({
17.       url: '../intro/intro?id=' + id
18.     })
19.   },
20.   /**
21.    * 生命周期函数--监听页面加载
22.    */
23.   onLoad: function(options) {
24.     //获取图书列表
25.     books.get({
26.       success: res => {
27.         //console.log(res.data)
28.         this.setData({
29.           bookList: res.data
30.         })
31.       }
32.     })
33.   },
34. })
```

2 图书详情页代码展示

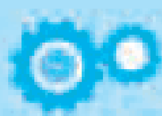
WXML 文件(pages/intro/intro.wxml)的完整代码如下：

```
1.  <!-- intro.wxml -->
2.  <!-- 下载时的蒙层 -->
3.  <view class='loading-container' wx:if='{{isDownloading}}'>
4.    <text>下载中,请稍候</text>
5.    <progress percent="{{percentNum}}" stroke-width="6" activeColor="#663366"
backgroundColor="#FFFFFF" show-info active active-mode="forwards"></progress>
6.  </view>
```

```
7.
8.  <!-- 图书详细信息 -->
9.  <view class='intro-container' wx:else>
10.   <!-- 图书封面 -->
11.   <image src='{{book.coverurl}}' mode='widthFix'></image>
12.
13.   <!-- 图书信息 -->
14.   <view class='intro-box'>
15.     <text>书名: {{book.title}}</text>
16.     <text>作者: {{book.author}}</text>
17.     <text>价格: {{book.price}}</text>
18.     <text>ISBN: {{book.isbn}}</text>
19.   </view>
20.
21.   <!-- "开始阅读"按钮 -->
22.   <button type='warn' bindtap='readBook'>开始阅读</button>
23. </view>
```

WXSS 文件(pages/intro/intro.wxss)的完整代码如下：

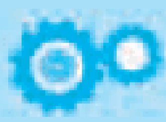
```
1.  /* 下载时的蒙层容器样式 */
2.  .loading-container {
3.    height: 100vh;                          /* 高度 */
4.    background-color: silver;               /* 背景颜色 */
5.    display: flex;                          /* flex 模型布局 */
6.    flex-direction: column;                 /* 水平排列 */
7.    align-items: center;                    /* 水平方向居中 */
8.    justify-content: space-around;          /* 分散布局 */
9.  }
10. /* 进度条样式 */
11. progress{
12.   width: 80%;                             /* 宽度 */
13. }
14. /* 图书详细信息区域 */
15. .intro-container{
16.   height: 100vh;
17.   display: flex;
18.   flex-direction: column;
19.   align-items: center;
20.   justify-content: space-evenly;
21. }
22. /* 图书封面图片 */
23. .intro-container image{
24.   width: 400rpx;
25.   margin: 20rpx;
26. }
27. /* 图书信息区域 */
28. .intro-box{
29.   display: flex;
30.   flex-direction: column;
31. }
32. /* 图书文字信息 */
33. .intro-box text{
34.   margin: 20rpx;
35. }
```



JS 文件(pages/intro/intro.js)的完整代码如下:

```
1.  const db = wx.cloud.database()
2.  const books = db.collection('books')
3.
4.  Page({
5.    /**
6.     * 页面的初始数据
7.     */
8.    data: {
9.      isDownloading: false,
10.     percentNum: 0
11.   },
12.   /**
13.    * 打开图书
14.    */
15.   openBook: function(path) {
16.     wx.openDocument({
17.       filePath: path,
18.       success: function(res) {
19.         console.log('打开图书成功')
20.       },
21.       fail: function(error) {
22.         console.log(error);
23.       }
24.     })
25.   },
26.   /**
27.    * 保存图书
28.    */
29.   saveBook: function(id, path) {
30.     var that = this
31.     wx.saveFile({
32.       tempFilePath: path,
33.       success: function(res) {
34.         //将文件地址存到本地缓存中,下次直接打开
35.         let newPath = res.savedFilePath
36.         wx.setStorageSync(id, newPath)
37.         //打开图书
38.         that.openBook(newPath)
39.       },
40.       fail: function(error) {
41.         console.log(error)
42.         that.showTips('文件保存失败!')
43.       }
44.     })
45.   },
46.   /**
47.    * 阅读图书
48.    */
49.   readBook: function(e) {
50.     var that = this
51.     //获取当前图书 ID
52.     let id = this.data.book._id
53.     //获取当前图书云端地址
54.     let fileid = this.data.book.fileid
```

```
55.     //查看本地缓存
56.     let path = wx.getStorageSync(id)
57.     //未曾下载过
58.     if (path == '') {
59.       //切换到下载时的蒙层
60.       that.setData({
61.         isDownloading: true
62.       })
63.       //先从云端下载图书
64.       const downloadTask = wx.cloud.downloadFile({
65.         fileID: fileid,
66.         success:res=>{
67.           //关闭下载时的蒙层
68.           this.setData({
69.             isDownloading: false
70.           })
71.           //下载成功
72.           if (res.statusCode == 200) {
73.             //获取地址
74.             path = res.tempFilePath
75.             //保存并打开图书
76.             this.saveBook(id, path)
77.           }
78.           //连上了服务器,下载失败
79.           else {
80.             this.showTips('暂时无法下载!')
81.           }
82.
83.         },
84.         //请求失败
85.         fail:err=> {
86.           //关闭下载时的蒙层
87.           this.setData({
88.             isDownloading: false
89.           })
90.           this.showTips('无法连接到服务器!')
91.         }
92.       })
93.
94.       //监听当前文件的下载进度
95.       downloadTask.onProgressUpdate(function(res) {
96.         let progress = res.progress;
97.         that.setData({
98.           percentNum: progress
99.         })
100.      })
101.    }
102.    //之前下载过的,直接打开
103.    else {
104.      //打开图书
105.      that.openBook(path)
106.    }
107.  },
108.
109.  /**
```



```
110.   * 封装 showModal 方法
111.   */
112.  showTips: function(content) {
113.    wx.showModal({
114.      title: '提醒',
115.      content: content,
116.      showCancel: false
117.    })
118.  },
119.
120.  /**
121.   * 生命周期函数--监听页面加载
122.   */
123.  onLoad: function(options) {
124.    //获取当前图书信息
125.    books.doc(options.id).get({
126.      success: res => {
127.        this.setData({
128.          book: res.data
129.        })
130.      }
131.    })
132.  },
133.})
```

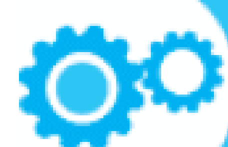
综合篇



第19章 Chapter 19

小程序云开发·基于全套云能力的图片分享社区

在学习了小程序云开发的基础知识后，读者不妨尝试综合应用云数据库、云函数和云存储制作一款图片分享社区小程序。用户可以上传图片到云存储空间中，也可以查看其他用户上传的图片列表，并进行图片分享、下载和全屏预览等。



本章学习目标

- 综合应用小程序云开发的基础知识创建图片分享社区小程序；
- 掌握云数据库创建、云存储管理和云函数调用等知识。

19.1 初始化项目



19.1.1 创建云模板项目



视频讲解

首先需要创建一个云开发项目，在任意盘符下创建一个空白文件夹(例如cloudPhoto)，然后填入AppID和选中"小程序·云开发"，如图19-1所示。

接着删除其中无用的模板代码：

(1) 删除cloudFunctions文件夹下的默认云函数login的全部内容。

(2) 找到app.json文件，打开并删除其中的页面引用，只保留第一个pages/index/index。

(3) 打开硬盘中的pages文件夹，删除index以外的所有目录。

(4) 进入index文件夹，删除多余的图片，以及JS、WXML和WXSS文件中的全部代码(JS文件删除干净后输入关键词page让它自动生成Page({})函数的相关代码)。

(5) 删除app.wxss中的代码。

(6) 删除image文件夹中的所有图片。

(7) 删除style文件夹，此时项目清理全部完成。

19.1.2 部署云数据库

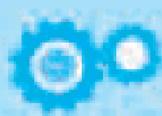


视频讲解

具体操作如下：

(1) 打开云开发控制台，创建一个新的数据集，例如photos。

(2) 检查photos数据集的权限，确认是"所有用户可读，仅创建者及管理



员可写”,完成。



图 19-1　云模板项目填写效果示意图

效果如图 19-2 所示。



图 19-2　在云开发控制台添加数据集 photos 并设置权限

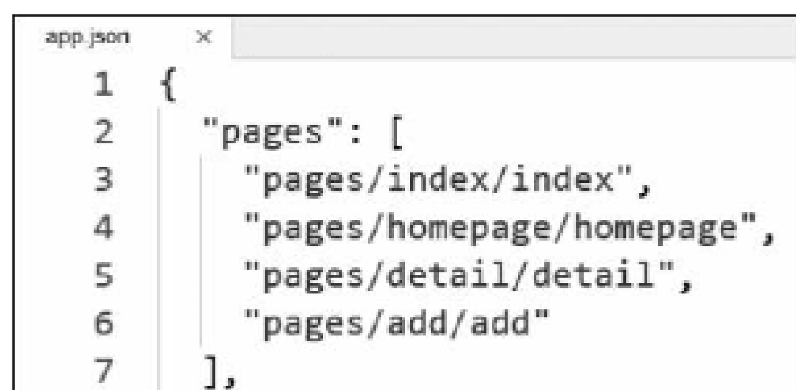
19.1.3　创建页面文件



视频讲解

本项目有 4 个页面文件，分别是 index（首页）、homepage（个人主页）、detail（图片展示页）和 add（上传图片页）。打开 app.json 文件，在 pages 属性中追加除 index 之外的其他页面的路径描述，如图 19-3 所示。

保存后会自动生成后面 3 个页面，此时 pages 文件夹内部的目录结构如图 19-4 所示。此时文件配置就全部完成，19.2 节将正式进行页面布局和样式设计。



```
app.json ×
1  {
2    "pages": [
3      "pages/index/index",
4      "pages/homepage/homepage",
5      "pages/detail/detail",
6      "pages/add/add"
7    ],
```

图 19-3　在 app.json 中添加页面配置的代码

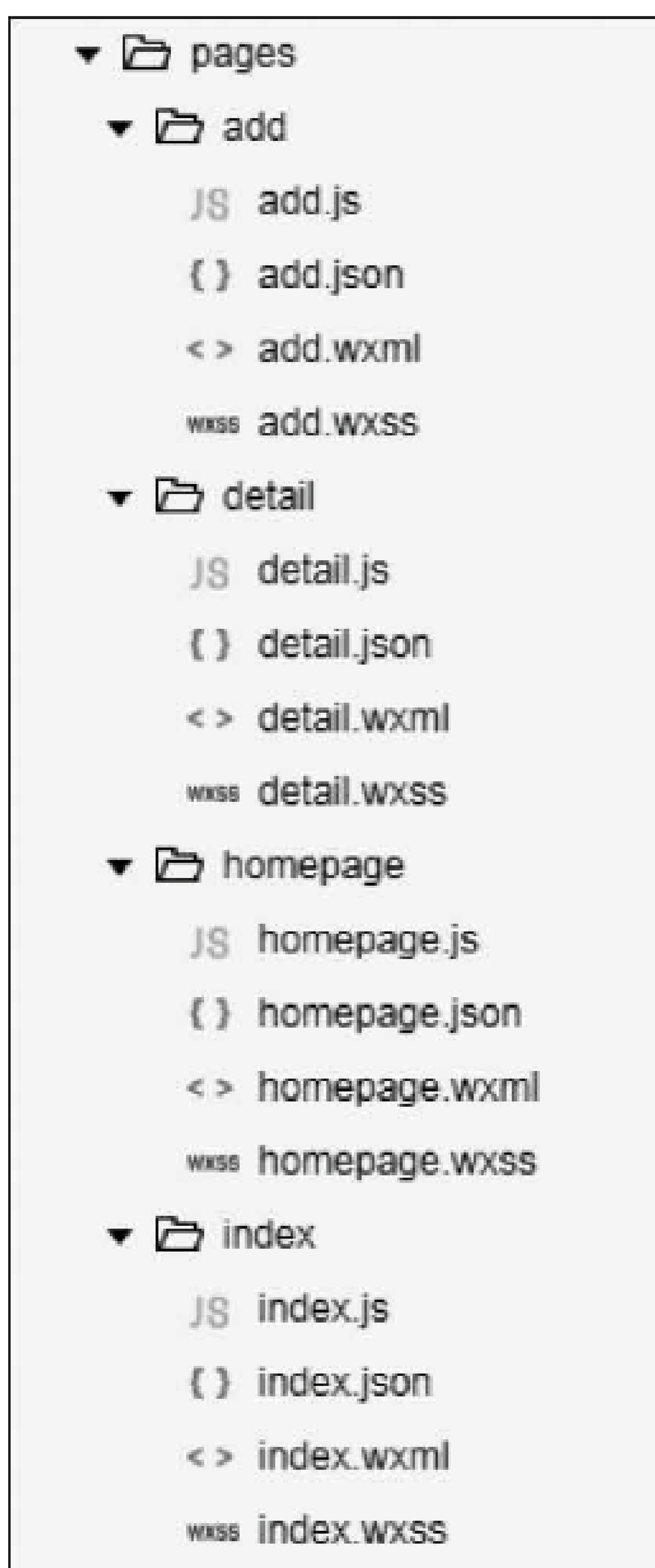
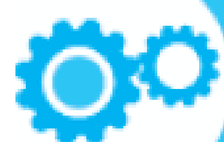


图 19-4　页面文件创建完成



19.2　视图设计



19.2.1　导航栏设计



视频讲解

用户可以通过在 app.json 中对 window 属性进行重新配置来自定义导航栏效果。

更改后的 app.json 文件代码如下：

```
1.  {
2.    "pages": [代码略],
3.    "window": {
4.      "navigationBarBackgroundColor": "#F6F6F6",
5.      "navigationBarTitleText": "图片分享社区",
6.      "navigationBarTextStyle": "black"
7.    }
8.  }
```

上述代码可以更改导航栏背景色为浅灰色、字体为黑色，效果如图 19-5 所示。

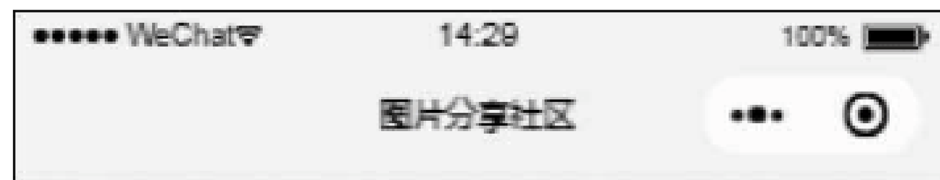


图 19-5　自定义导航栏效果

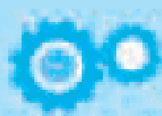
19.2.2　页面设计



视频讲解

1 首页设计

首页主要是图片展示区域，用于展示多名作者上传的图片作品。每张图



片都以卡片的形式出现，卡片包括页眉、主体和页脚部分。除此之外还需要在右上角放置一个浮动按钮，点击此按钮可以上传图片到云存储空间。

页面设计如图 19-6 所示。

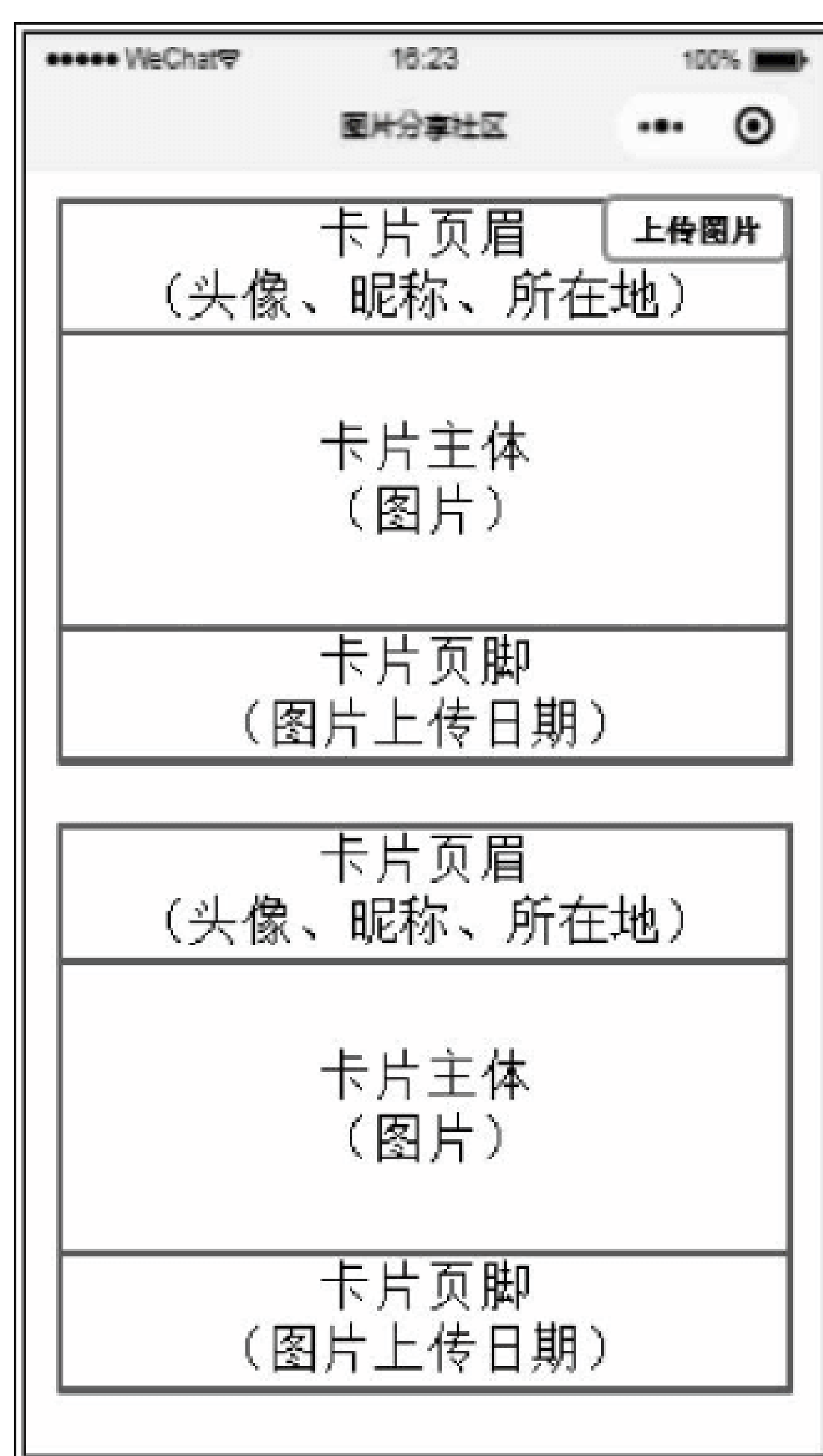


图 19-6　首页设计图

计划使用如下组件。

- 整体容器：容器组件（< view >），并定义 class＝'container'；
- 整张卡片：容器组件（< view >），并定义 class＝'card'；
- 卡片页眉、主体和页脚：容器组件（< view >），并分别定义 class＝'card-head'、'card-body'和'card-foot'；
- 作者头像图标和上传的图片：图片组件（< image >）；
- 作者昵称、所在地以及图中上传时间：文本组件（< text >）；
- 浮动按钮：按钮组件（< button >）。

WXML（pages/index/index.wxml）代码如下：

```
1.  < view class = 'container'>
2.    <!-- 卡片 -->
3.    < view class = 'card'>
4.      <!-- 卡片页眉 -->
5.      < view class = 'card - head'>
6.        < image class = 'avatar' src = '/images/avatar.jpg' mode = 'widthFix'></image >
7.        < view class = 'title'>
8.          < text class = 'nickName'>周小贝</text >
9.          < text class = 'address'> Anhui,China </text >
10.       </view >
11.     </view >
12.
13.     <!-- 卡片主体 -->
14.     < view class = 'card - body'>
15.       < image src = '/images/pic001.jpg' mode = 'widthFix'></image >
16.     </view >
17.
18.     <!-- 卡片页脚 -->
19.     < view class = 'card - foot'>
20.       < text > 2019 - 06 - 21 </text >
21.     </view >
22.   </view >
23. </view >
24.
25. <!-- "上传图片"按钮 -->
26. < view class = 'floatBtn'>
27.   < button size = 'mini' type = 'primary'>上传图片</button >
28. </view >
```

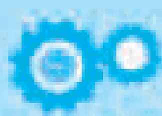
由于整体容器、卡片样式也会在其他页面使用，所以将这两部分样式代码写到公共样式表 app.wxss 中，具体代码如下：

```
1.  /* 1.整体容器样式 */
2.  .container {
3.    display: flex;
4.    flex - direction: column;
```

```
5.     align - items: center;
6.     background - color: whitesmoke;
7.     min - height: 100vh;
8.   }
9.   /*2.卡片区域整体样式*/
10.  .card {
11.    width: 710rpx;
12.    border - radius: 20rpx;
13.    background - color: white;
14.    margin: 20rpx 0;
15.  }
16.  /*2-1 卡片页眉*/
17.  .card - head {
18.    display: flex;
19.    flex - direction: row;
20.    align - items: center;
21.  }
22.
23.  /*2-1-1 头像图片*/
24.  .avatar {
25.    width: 150rpx;
26.    border - radius: 50%;
27.    margin: 0 20rpx;
28.  }
29.
30.  /*2-1-2 文字信息*/
31.  .title {
32.    display: flex;
33.    flex - direction: column;
34.  }
35.  /*2-1-2(1) 昵称*/
36.  .nickName {
37.    font - size: 35rpx;
38.    margin: 0 10rpx;
39.  }
40.  /*2-1-2(2) 所在地区*/
41.  .address {
42.    font - size: 30rpx;
43.    color: gray;
44.    margin: 0 10rpx;
45.  }
46.  /*2-2 卡片主体*/
47.  .card - body image {
48.    width: 100%;
49.  }
50.  /*2-3 卡片页脚*/
51.  .card - foot{
52.    font - size: 35rpx;
53.    text - align: right;
54.    margin: 20rpx;
55.  }
```

浮动按钮是首页特有的,因此将相关样式代码写到首页对应的 index.wxss 文件中即可。

WXSS(pages/index/index.wxss)代码如下:



```
1.  /* 浮动按钮样式 */
2.  .floatBtn{
3.    position: fixed;
4.    top: 50rpx;
5.    right: 50rpx;
6.  }
```

上述代码表示无论首页如何下拉滚动,浮动按钮都永远固定在距离顶端和右边 50rpx 的位置上不变。当前效果如图 19-7 所示。

需要注意的是,这里所展示的头像、昵称、所在地、图片和上传日期都是为了演示效果临时填写的,后续会换成实际录入的信息,并形成卡片列表效果。



视频讲解

2 个人主页设计

个人主页除了顶部会显示当前作者的头像、昵称外,下方仍然是和首页一样布局的图片展示区域,仅限于展示当前作者上传的图片作品。

页面设计如图 19-8 所示。



图 19-7 首页效果图



图 19-8 个人主页设计图

计划使用如下组件。

- 整体容器:容器组件(<view>),并定义 class='container';
- 顶端头像和昵称区域:容器组件(<view>),并定义 class='avatarBox';
- 整张卡片:容器组件(<view>),并定义 class='card';
- 卡片页眉、主体和页脚:容器组件(<view>),并分别定义 class='card-head'、'card-body'和'card-foot';
- 作者头像图标和上传的图片:图片组件(<image>);
- 作者昵称和上传时间:文本组件(<text>)。

WXML(pages/homepage/homepage.wxml)代码如下:

```
1.  <view class='container'>
2.    <!-- 头像和昵称 -->
3.    <view class='avatarBox'>
4.      <image src='/images/avatar.jpg'></image>
5.      <text>周小贝个人主页</text>
6.    </view>
7.
8.    <!-- 卡片 -->
9.    <view class='card'>
10.     <!-- 卡片页眉 -->
11.     <view class='card-head'>
12.       <image class='avatar' src='/images/avatar.jpg' mode='widthFix'></image>
13.       <view class='title'>
14.         <text class='nickName'>周小贝</text>
15.         <text class='address'>Anhui,China</text>
16.       </view>
17.     </view>
18.
19.     <!-- 卡片主体 -->
20.     <view class='card-body'>
21.       <image src='/images/pic01.jpg' mode='widthFix'></image>
22.     </view>
23.
24.     <!-- 卡片页脚 -->
25.     <view class='card-foot'>
26.       <text>2019-06-21</text>
27.     </view>
28.   </view>
29. </view>
```

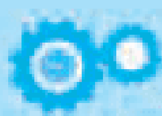
整体容器和卡片的样式代码与首页相同，可以自动继承于公共样式表 app.wxss，无须重复声明，而顶部头像和昵称的相关样式写到对应的 homepage.wxss 中即可。

WXSS(pages/homepage/homepage.wxss)代码如下：

```
1.  /*头像区域*/
2.  .avatarBox{
3.    display: flex;
4.    flex-direction: column;
5.    align-items: center;
6.    padding-top: 100rpx;
7.  }
8.  /*头像区域中的头像图片*/
9.  .avatarBox image{
10.   width: 200rpx;
11.   height: 200rpx;
12.   border-radius: 50%;
13. }
14. /*头像区域中的昵称文本*/
15. .avatarBox text{
16.   margin: 30rpx 0;
17. }
```

当前效果如图 19-9 所示。

同样这里所展示的头像、昵称、图片以及上传日期都是为了演示效果临时填写的，后续会换成实际录入的信息，并形成卡片列表效果。



3 图片展示页设计



视频讲解

当用户点击卡片主体中的图片时会跳转到图片展示页查看完整图片效果，该页面主要包含完整图片和 3 个按钮，按钮分别用于下载图片、分享给好友以及全屏预览图片。

页面设计如图 19-10 所示。

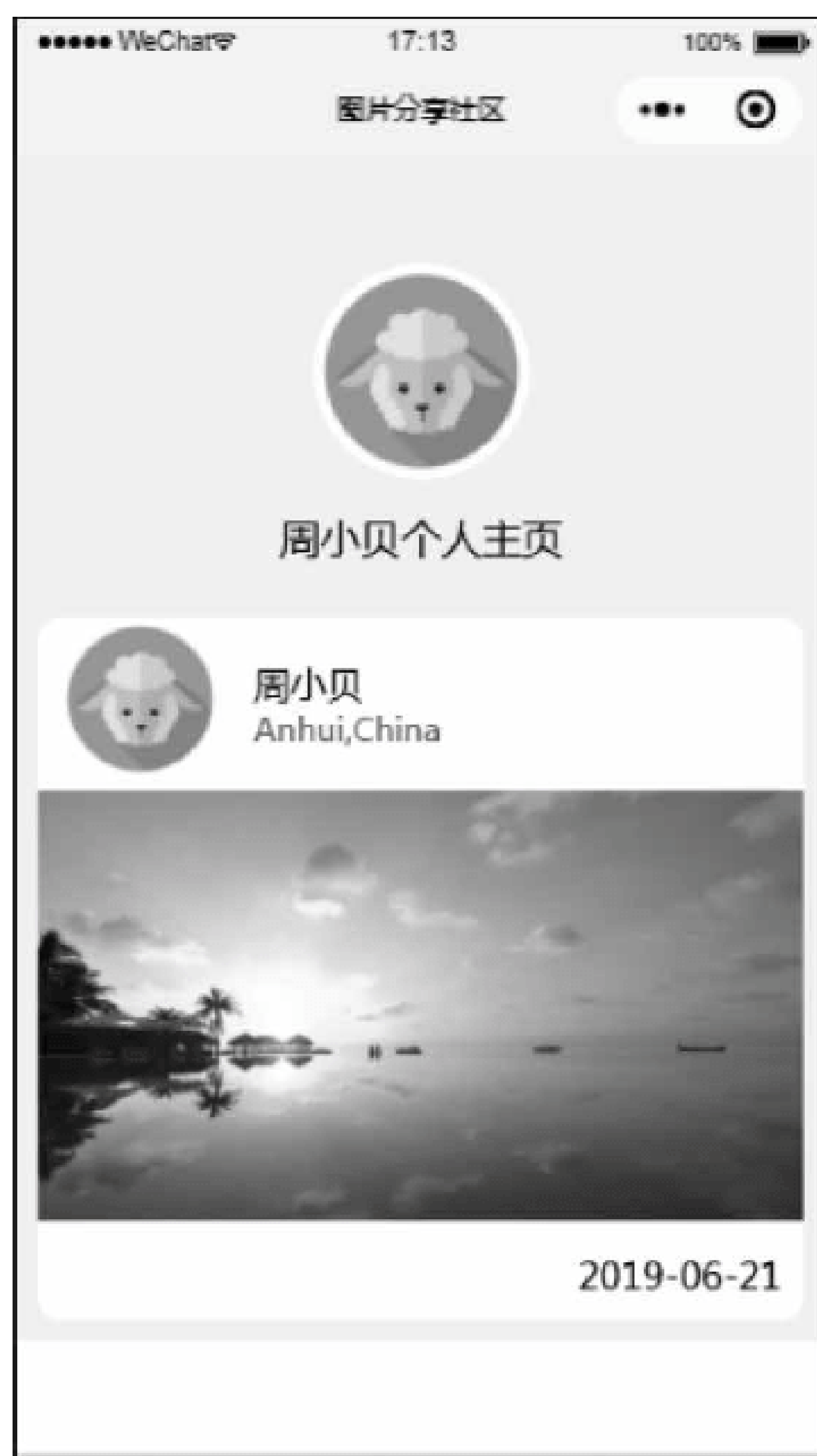


图 19-9　个人主页效果图

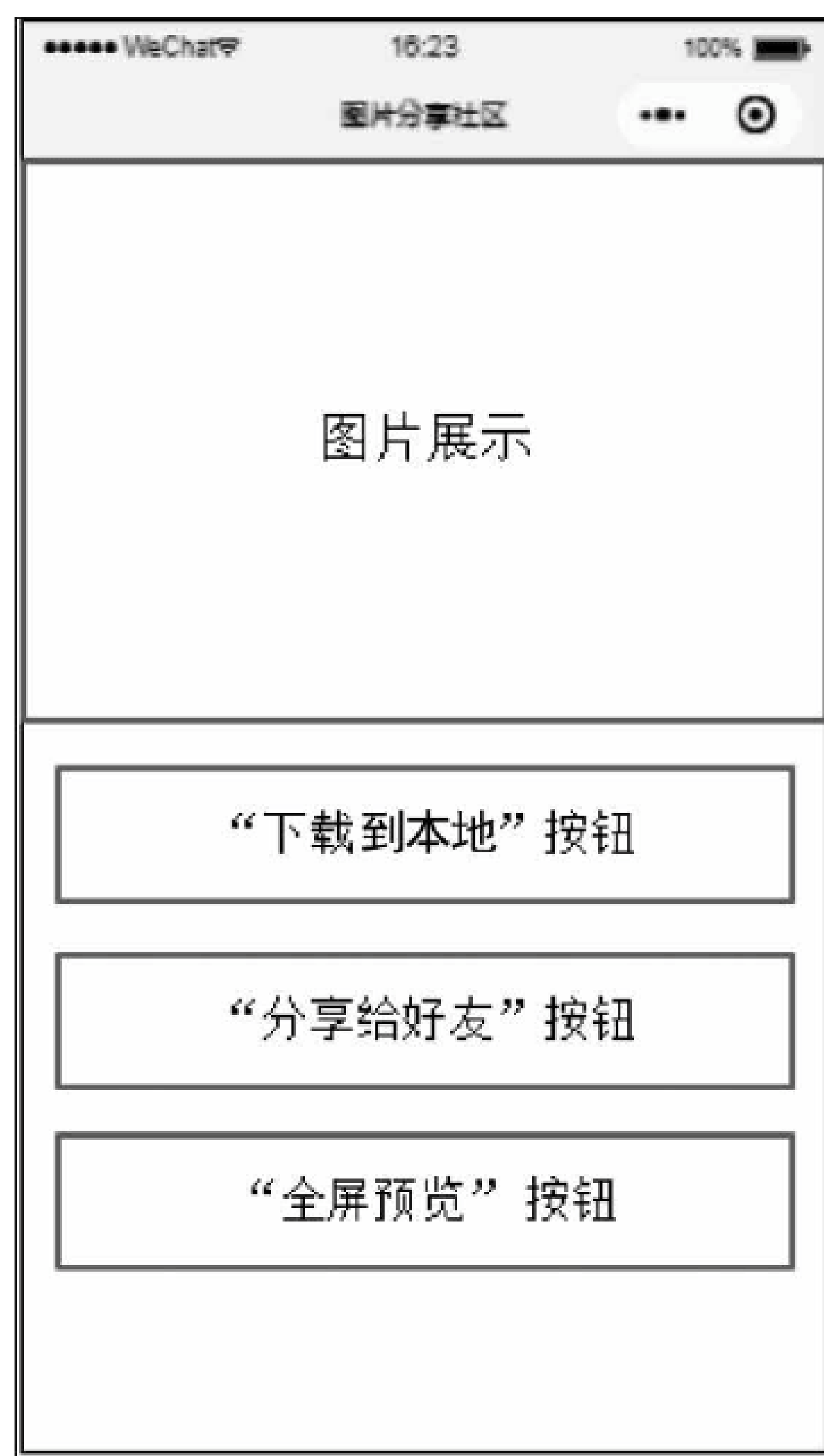


图 19-10　图片展示页设计图

计划使用如下组件。

- 顶端图片展示：图片组件(<image>)；
- 按钮区域：容器组件(<view>)；
- 3 个按钮：按钮组件(<button>)。

WXML 文件(pages/detail/detail. wxml)代码如下：

```
1.  <!-- 顶端图片展示 -->
2.  <image src = '/images/pic01.jpg' mode = 'widthFix'></image>
3.
4.  <!-- 按钮区域 -->
5.  <view>
6.    <button type = 'primary' plain>下载到本地</button>
7.    <button type = 'primary' plain>分享给好友</button>
8.    <button type = 'primary' plain>全屏预览</button>
9.  </view>
```

WXSS(pages/detail/detail. wxss)代码如下：

```
1.  /* 顶端展示图片样式 */
2.  image{
3.    width: 750rpx;
4.  }
5.
```

```
6.  /* 按钮外层 view */
7.  view{
8.    padding: 0 50rpx;
9.  }
10.
11. /* 按钮样式 */
12. button{
13.   margin: 10rpx;
14. }
```

当前效果如图 19-11 所示。

这里的图片路径是为了演示效果临时填写的，后续会换成实际需要展示的图片。



视频讲解

4 上传图片页设计

点击首页中的“上传图片”按钮可以跳转到上传图片页，该页面主要包含按钮、标题和九宫格图片。页面设计如图 19-12 所示。



图 19-11 图片展示页效果图

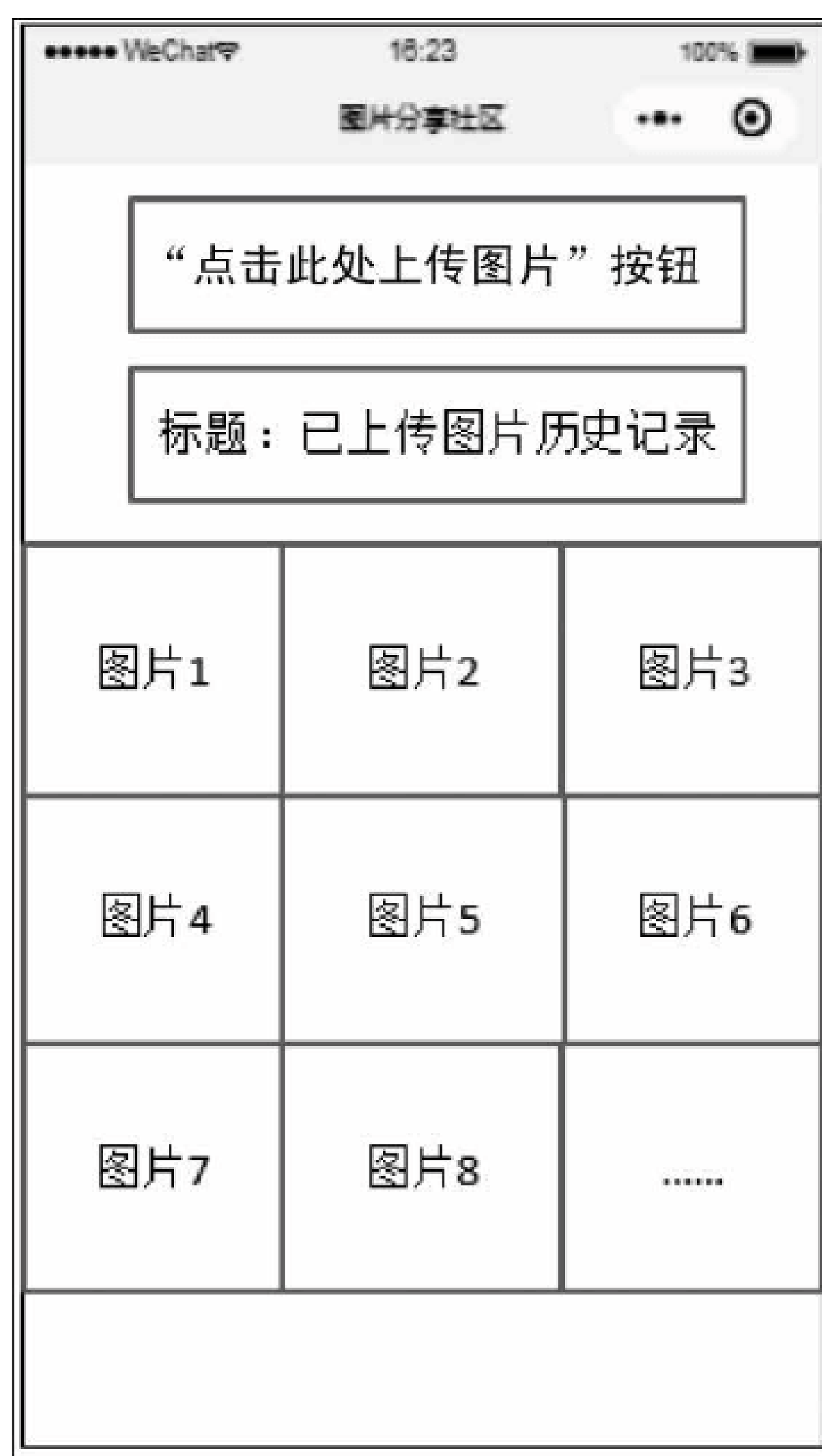


图 19-12 上传图片页设计图

计划使用如下组件。

- 整体容器：容器组件(<view>)，并定义 class='container'；
- 按钮：按钮组件(<button>)；
- 标题：文本组件(<text>)；
- 九宫格图片展示：图片组件(<image>)。

WXML 文件(pages/add/add.wxml)代码如下：

```
1. <view class = 'container'>
2.   <!-- “点击此处上传图片”按钮 -->
3.   <button type = 'warn'>点击此处上传图片</button>
4.
```

```
5.     <!-- 已上传图片区域 -->
6.     <view class='photoBox'>
7.       <!-- 标题 -->
8.       <text>已上传图片历史记录</text>
9.
10.      <!-- 图片集 -->
11.      <view>
12.        <image src='/images/pic01.jpg'></image>
13.        <image src='/images/pic01.jpg'></image>
14.        <image src='/images/pic01.jpg'></image>
15.        <image src='/images/pic01.jpg'></image>
16.        <image src='/images/pic01.jpg'></image>
17.        <image src='/images/pic01.jpg'></image>
18.      </view>
19.    </view>
20.  </view>
```

整体容器的样式代码大部分与首页相同，可以自动继承于公共样式表 app.wxss，无须重复声明。在对应的 add.wxss 文件中还可以追加其他样式代码。

WXSS(pages/add/add.wxss)代码如下：

```
1.  /* 已上传图片区域 */
2.  .photoBox{
3.    margin-top: 50rpx;
4.    display: flex;
5.    flex-direction: column;
6.  }
7.  /* 标题样式 */
8.  .photoBox text{
9.    margin-bottom: 30rpx;
10.   text-align: center;
11. }
12. /* 图片样式 */
13. .photoBox image{
14.   float: left;
15.   width: 250rpx;
16.   height: 250rpx;
17. }
```

当前效果如图 19-13 所示。

为了体现多张图片设计效果，这里重复了 6 次<image>组件的相关代码，仅作为临时参考。后续将读取用户真实上传的图片数量，展示历史记录。

此时页面布局与样式设计就已完成，19.3 节将介绍如何进行逻辑处理。

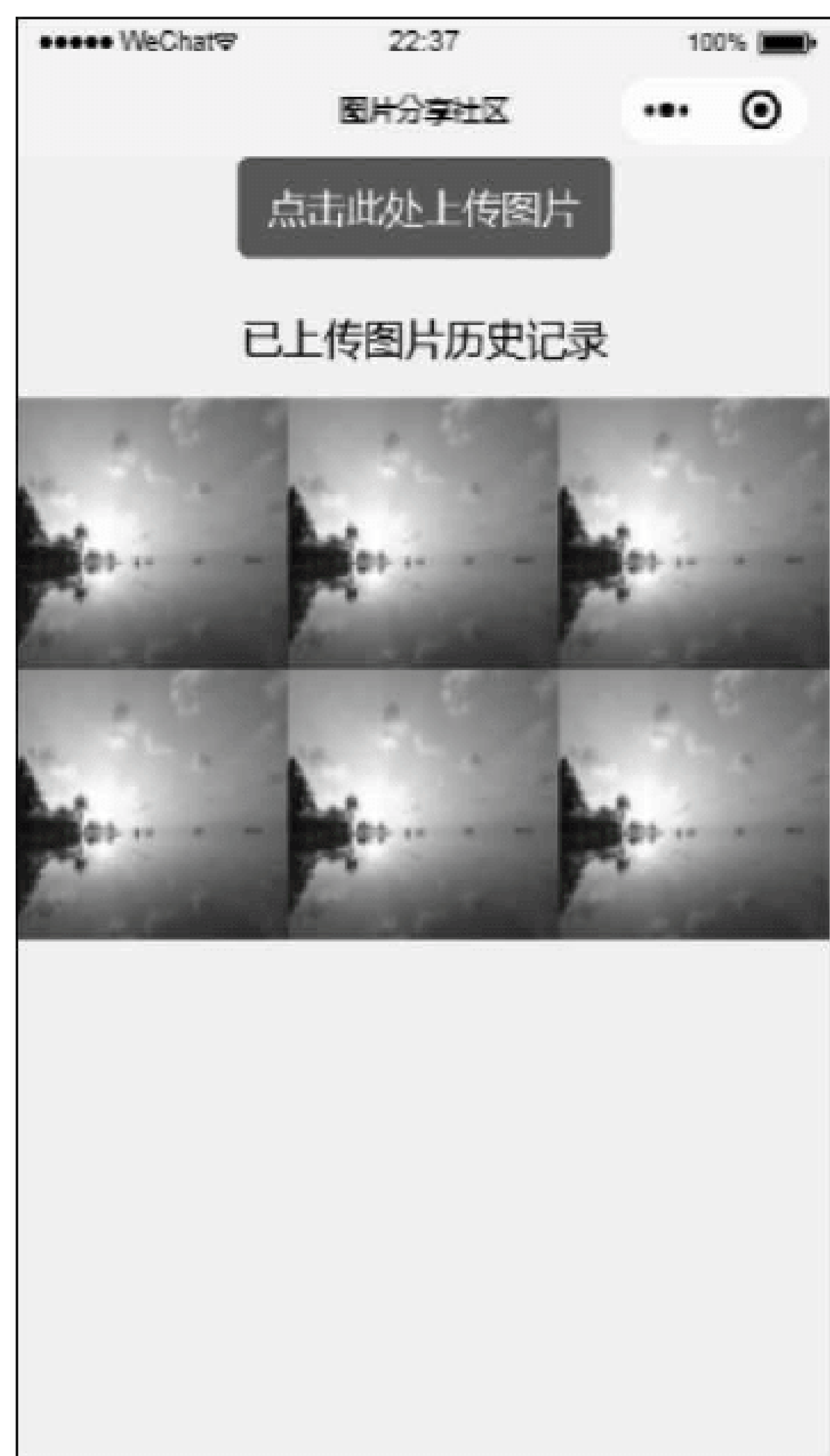
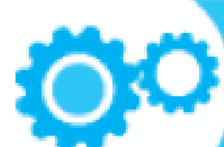


图 19-13　上传图片页效果图

 19.3　逻辑实现 

19.3.1　用户个人信息获取逻辑

在首页中点击“上传图片”按钮时需要实现两个功能，一是获取当前用户个人信息(包括基础信息和 openid)；二是能够跳转到上传图片页。

1 获取用户基础信息



视频讲解

用户基础信息主要包括用户的微信昵称、头像、性别、所在地等内容，为按钮组件(< button >)添加 open-type 属性就可以用来获取这些信息。

修改 index.wxml 文件中浮动按钮的代码，为其添加点击后获取用户个人信息的功能。

相关 WXML(pages/index/index.wxml)代码片段如下：

```
1.  <!-- "上传图片"按钮 -->
2.  <view class = 'floatBtn'>
3.    <button size = 'mini' type = 'primary' open - type = 'getUserInfo' bindgetuserinfo =
'getUserInfo'>上传图片</button>
4.  </view>
```

其中 open-type 的属性值为固定写法，用于表示按钮的特殊用法类型；bindgetuserinfo 的属性值为自定义函数，用于获取到用户信息后的进一步处理，开发者也可以更改为其他名称。

在 JS 文件中添加自定义函数 getUserInfo，相关 JS(pages/index/index.js)代码片段如下：

```
1.  var app = getApp()
2.
3.  Page({
4.    /**
5.     * 自定义函数 -- 获取用户个人信息
6.     */
7.    getUserInfo: function(e) {
8.      //尝试打印输出个人信息,测试是否获取成功
9.      console.log(e.detail.userInfo)
10.     //将用户个人信息存放到全局变量 userInfo 中
11.     app.globalData.userInfo = e.detail.userInfo
12.   },
13. })
```

点击右上角的按钮触发获取用户信息事件，此时 Console 控制台的运行效果如图 19-14 所示。

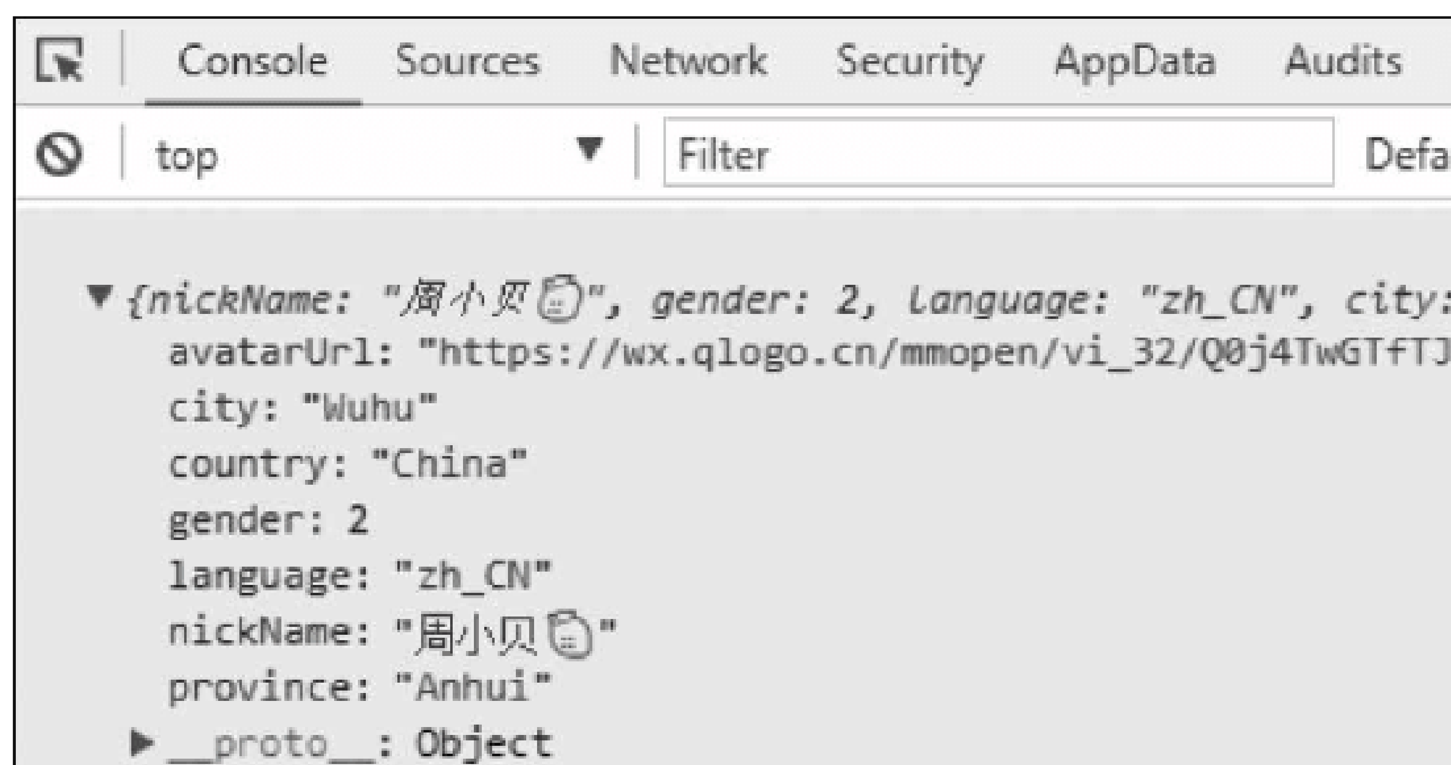


图 19-14　Console 控制台显示用户基础信息

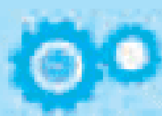
由图可见，点击按钮后可以将用户的信息打印出来了，说明该功能生效。此时可以去掉或者注释掉其中的 console.log(e.detail.userInfo)语句，该语句仅在开发过程中临时使用，用于测试是否获取到数据。



视频讲解

2 获取用户 openid 信息

实际上每个用户还有一个专属用户编号，称为 openid，该编号信息是不会



改变的，因此开发者可以通过 openid 的值来确定用户身份。

可以通过自定义云函数来实现用户 openid 信息的获取。对项目中的 cloudfunctions 右击，选择"新建 Node.js 云函数"，然后自定义云函数名称，例如 getOpenid。

修改云函数中 index.js 文件的代码如下：

```
1.  //云函数入口文件
2.  const cloud = require('wx-server-sdk')
3.
4.  cloud.init()
5.
6.  //云函数入口函数
7.  exports.main = async (event, context) => {
8.    const wxContext = cloud.getWXContext()
9.
10.   return {
11.     openid: wxContext.OPENID
12.   }
13. }
```

修改完毕后右击该云函数，选择"上传并部署：云端安装依赖"等待上传到云开发控制台。上传成功后可以在云开发控制台查看，如图 19-15 所示。

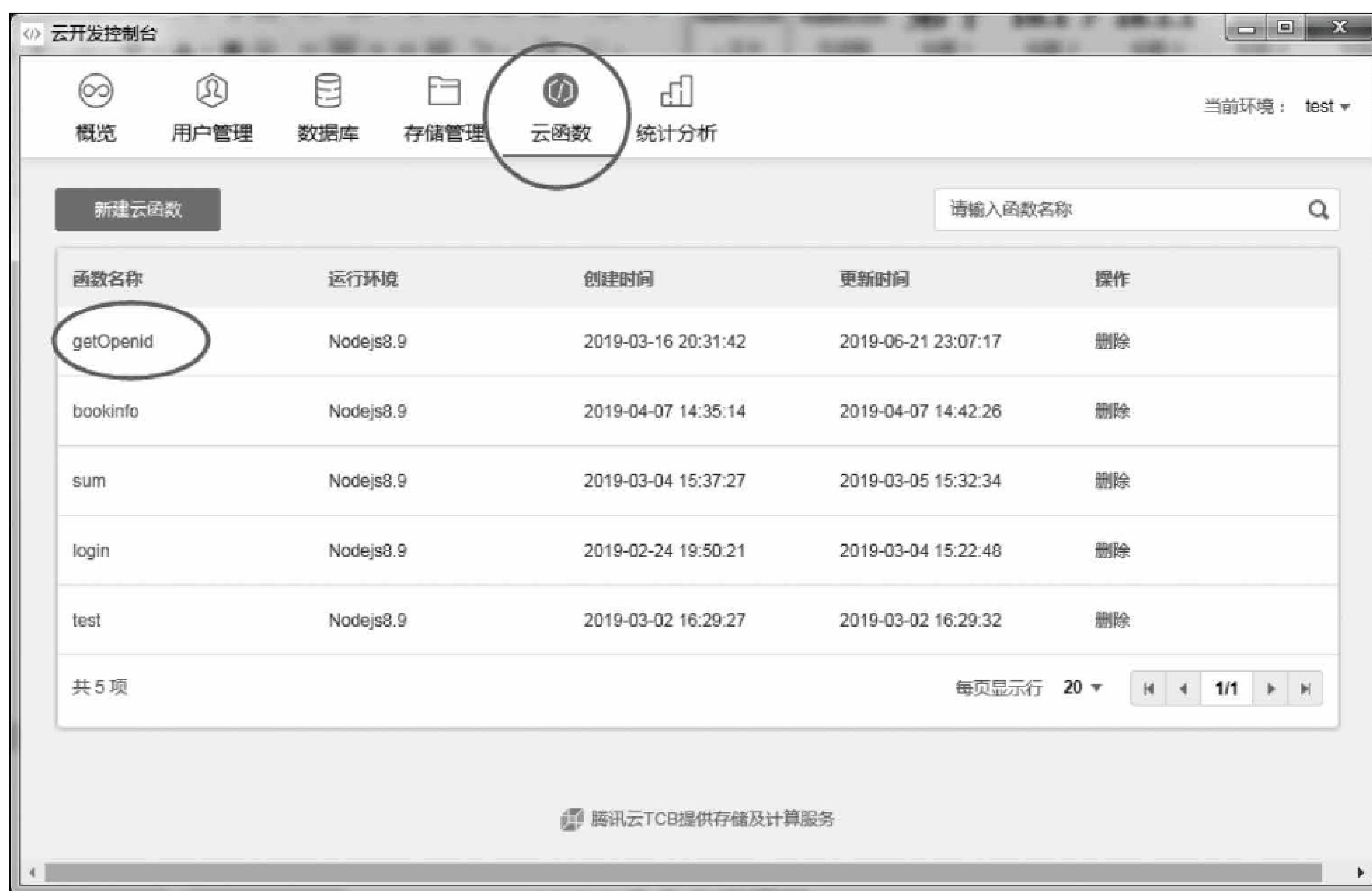


图 19-15　云开发控制台显示云函数信息

现在就可以在小程序中调用该云函数获取当前用户的 openid 信息了。修改 index.js 中的自定义函数 getUserInfo，相关 JS(pages/index/index.js)代码片段如下：

```
1.  Page({
2.    /**
3.     * 自定义函数 -- 获取用户个人信息
4.     */
5.    getUserInfo: function(e) {
```

```
6.        //尝试打印输出个人信息，测试是否获取成功
7.        //console.log(e.detail.userInfo)
8.        //将用户个人信息存放到全局变量 userInfo 中
9.        app.globalData.userInfo = e.detail.userInfo
10.
11.       //检测是否已经获取过了用户 openid 信息
12.       if (app.globalData.openid == null) {
13.         //如果是第一次登录则使用云函数获取用户 openid
14.         wx.cloud.callFunction({
15.           name: 'getOpenid',
16.           complete: res => {
17.             //尝试打印输出个人信息，测试是否获取成功
18.             console.log(res.result.openid)
19.             //将用户 openid 信息存放到全局变量 openid 中
20.             app.globalData.openid = res.result.openid
21.           }
22.         })
23.       }
24.     },
25.   })
```

点击右上角的按钮触发获取用户信息事件，此时 Console 控制台的运行效果如图 19-16 所示。

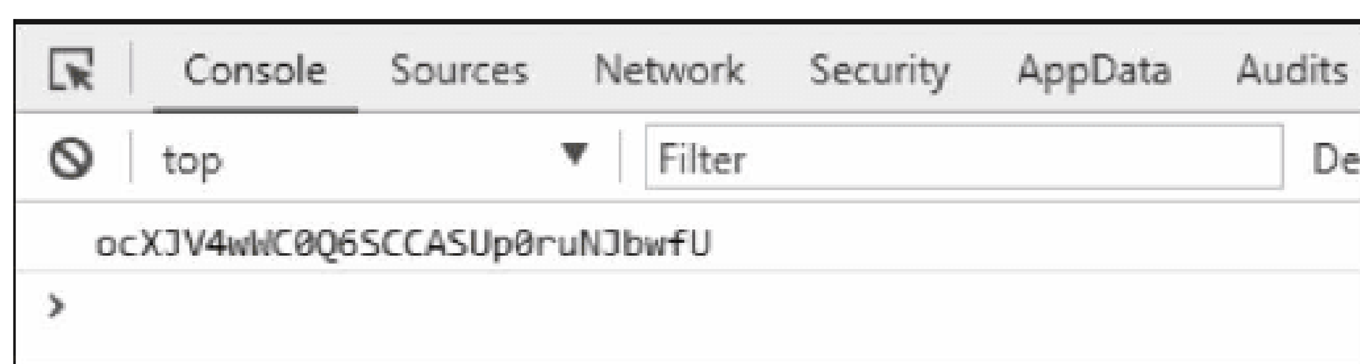


图 19-16 Console 控制台显示用户 openid 信息

由该图可见，点击按钮后可以将用户的 openid 信息打印出来了，说明该功能生效。此时可以去掉或者注释掉其中的 console.log(res.result.openid)语句，该语句仅在开发过程中临时使用，用于测试是否获取到数据。

3 首页跳转上传图片页

首页中的“上传图片”按钮点击后除了能够获取用户个人信息以外，还需要跳转到上传图片页，以便进行本地图片的上传。



视频讲解

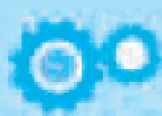
修改 index.wxml 文件中浮动按钮的代码，为其添加自定义点击事件 goToAdd。

相关 WXML(pages/index/index.wxml)代码片段如下：

```
1.  <!-- “上传图片”按钮 -->
2.  <view class = 'floatBtn'>
3.     < button size = 'mini' type = 'primary' bindtap = 'goToAdd' open - type = 'getUserInfo' bindgetuserinfo = 'getUserInfo'>上传图片</button>
4.  </view>
```

在 JS 文件中添加自定义函数 goToAdd，相关 JS(pages/index/index.js)代码片段如下：

```
1.  Page({
2.    /**
3.     * 自定义函数--跳转上传图片页
4.     */
```



```
5.     goToAdd: function(options) {
6.       wx.navigateTo({
7.         url: '../add/add',
8.       })
9.     },
10. })
```

为了测试跳转后新页面是否也可以读取到用户个人信息数据，可以临时在 add.js 文件的 onLoad 函数中添加 console.log()语句尝试打印输出用户的基础信息和 openid。

相关 JS(pages/add/add.js)代码片段如下：

```
1.  var app = getApp()
2.  Page({
3.    /**
4.     * 生命周期函数--监听页面加载
5.     */
6.    onLoad: function(options) {
7.      //尝试输出用户基础信息
8.      console.log(app.globalData.userInfo)
9.      //尝试输出用户 openid
10.     console.log(app.globalData.openid)
11.   },
12. })
```

点击首页右上角的按钮跳转新页面，此时 Console 控制台的运行效果如图 19-17 所示。

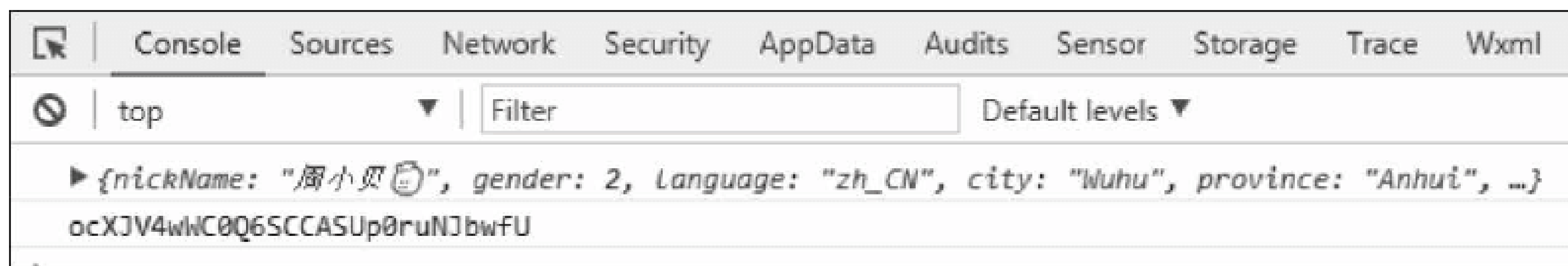


图 19-17　Console 控制台显示用户个人信息

由图可见，点击按钮跳转上传图片页后可以将用户基础信息和 openid 都打印出来了，说明该功能生效。此时可以去掉或者注释掉这两个 console.log()语句，该语句仅在开发过程中临时使用，用于测试是否获取到数据。

19.3.2　上传图片页逻辑

为了使首页可以显示真实上传的图片素材，可以先将上传图片页的功能实现，然后传若干张测试图片等待首页展示使用。

1 图片上传

图片上传成功后需要用到云数据库中的 photos 数据集记录图片基础信息，例如图片地址、添加日期、上传者信息等。因此首先在 add.js 文件顶端声明对 photos 数据集的引用，相关 JS(pages/index/index.js)代码片段如下：



视频讲解

```
1.  const db = wx.cloud.database()
2.  const photos = db.collection('photos')
3.
4.  Page({
5.      …
6.  })
```

其中 const photos 是自定义名称，开发者可以自行更改。

上传图片成功后还需要记录上传日期，因此在 add. js 文件的顶端自定义函数 formatDate，用于格式化显示当天的日期。相关 JS(pages/index/index. js)代码片段如下：

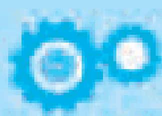
```
1.  //格式化当前日期
2.  function formatDate() {
3.    var now = new Date()
4.    var year = now.getFullYear()
5.    var month = now.getMonth() + 1
6.    var day = now.getDate()
7.
8.    if (month < 10) month = '0' + month
9.    if (day < 10) day = '0' + day
10.
11.   return year + '-' + month + '-' + day
12. }
13.
14. Page({
15.     …
16. })
```

修改 add. wxml 中按钮的代码，为其添加自定义点击事件 upload，用于上传图片。相关 WXML(pages/add/add. wxml)代码片段修改后如下：

```
1.    <!--"点击此处上传图片"按钮 -->
2.    <button type='warn' bindtap='upload'>点击此处上传图片</button>
```

然后在 add. js 中添加自定义函数 upload，相关 JS(pages/add/add. js)代码片段如下：

```
1.  Page({
2.    /**
3.     * 自定义函数--上传图片
4.     */
5.    upload: function() {
6.      //选择图片
7.      wx.chooseImage({
8.        count: 1,
9.        sizeType: ['compressed'],
10.       sourceType: ['album', 'camera'],
11.       success: function(res) {
12.         //loading 提示框表示正在上传图片
13.         wx.showLoading({
14.           title: '上传中',
15.         })
16.         //获取图片临时地址
17.         const filePath = res.tempFilePaths[0]
18.
19.         //自定义云端的图片名称
20.         const cloudPath = Math.floor(Math.random() * 1000000) + filePath.match(/.[^.]
+?$/)[0]
21.         //上传图片到云存储空间中
22.         wx.cloud.uploadFile({
23.           cloudPath,
24.           filePath,
25.           success: res => {
```



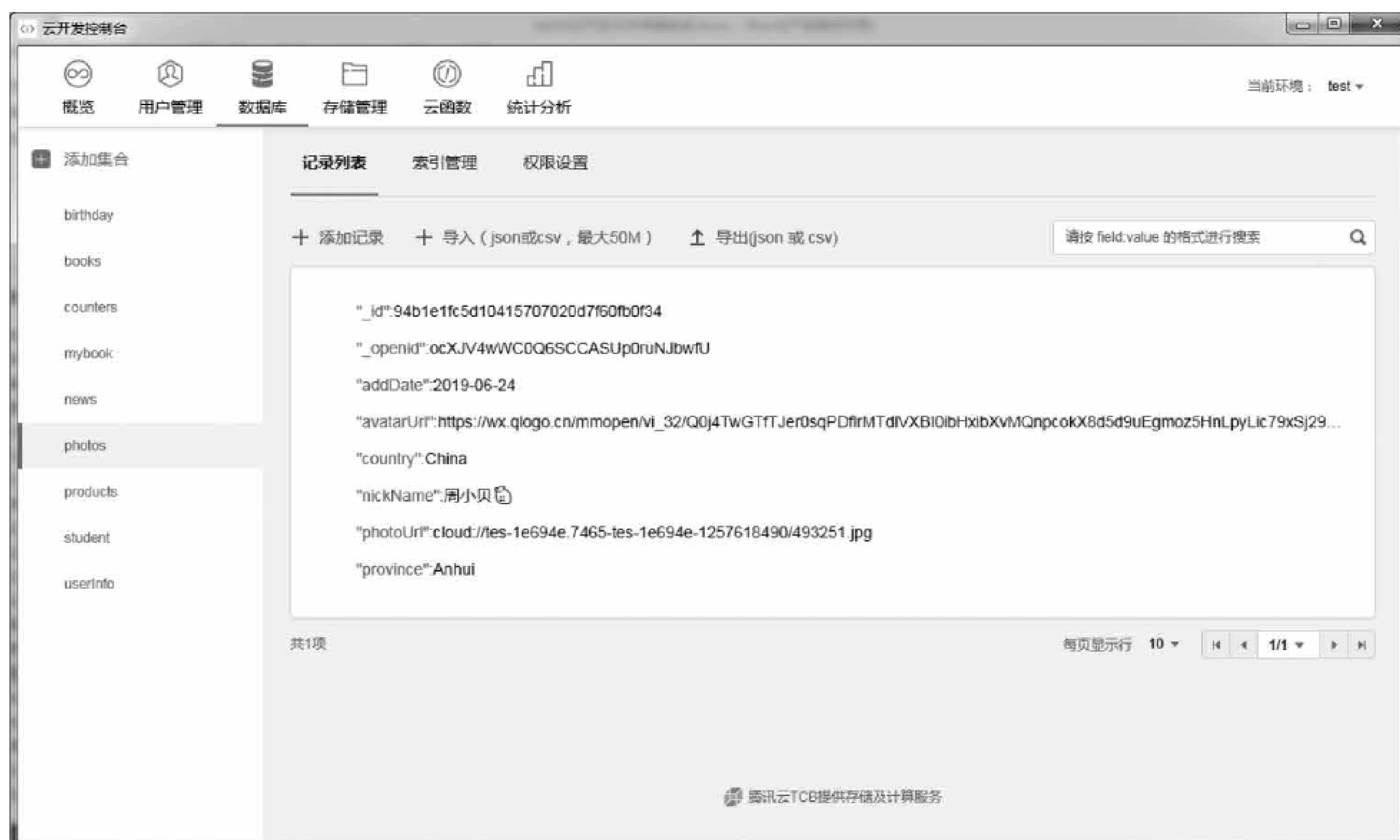
```
26.                 //提示上传成功
27.                 wx.showToast({
28.                   title: '上传成功!',
29.                   duration: 3000
30.                 })
31.
32.                 //获取用户个人基础信息
33.                 let userInfo = app.globalData.userInfo
34.                 //获取当天日期
35.                 let today = formatDate()
36.
37.                 //往云数据集中添加一条记录
38.                 photos.add({
39.                   data: {
40.                     photoUrl: res.fileID,
41.                     avatarUrl: userInfo.avatarUrl,
42.                     country: userInfo.country,
43.                     province: userInfo.province,
44.                     nickName: userInfo.nickName,
45.                     addDate: today
46.                   },
47.                   success: res => {
48.                     console.log(res)
49.                   },
50.                   fail: e => {
51.                     console.log(e)
52.                   }
53.                 })
54.               },
55.               fail: e => {
56.                 //提示上传失败
57.                 wx.showToast({
58.                   icon: 'none',
59.                   title: '上传失败',
60.                 })
61.               }
62.             })
63.           },
64.           fail: e => {
65.             console.error(e)
66.           },
67.           complete: () => {
68.             //上传完成后关闭 loading 提示框
69.             wx.hideLoading()
70.           }
71.         })
72.       },
73.     })
```

上述代码在自定义云端图片名称时用了随机数 Math.floor(Math.random() * 1000000)，表示图片名称为随机生成的 0～1 000 000 的向下取整的数，但是在图片非常多的情况下难免会碰巧有重复的，开发者也可以思考其他自定义的图片名称。

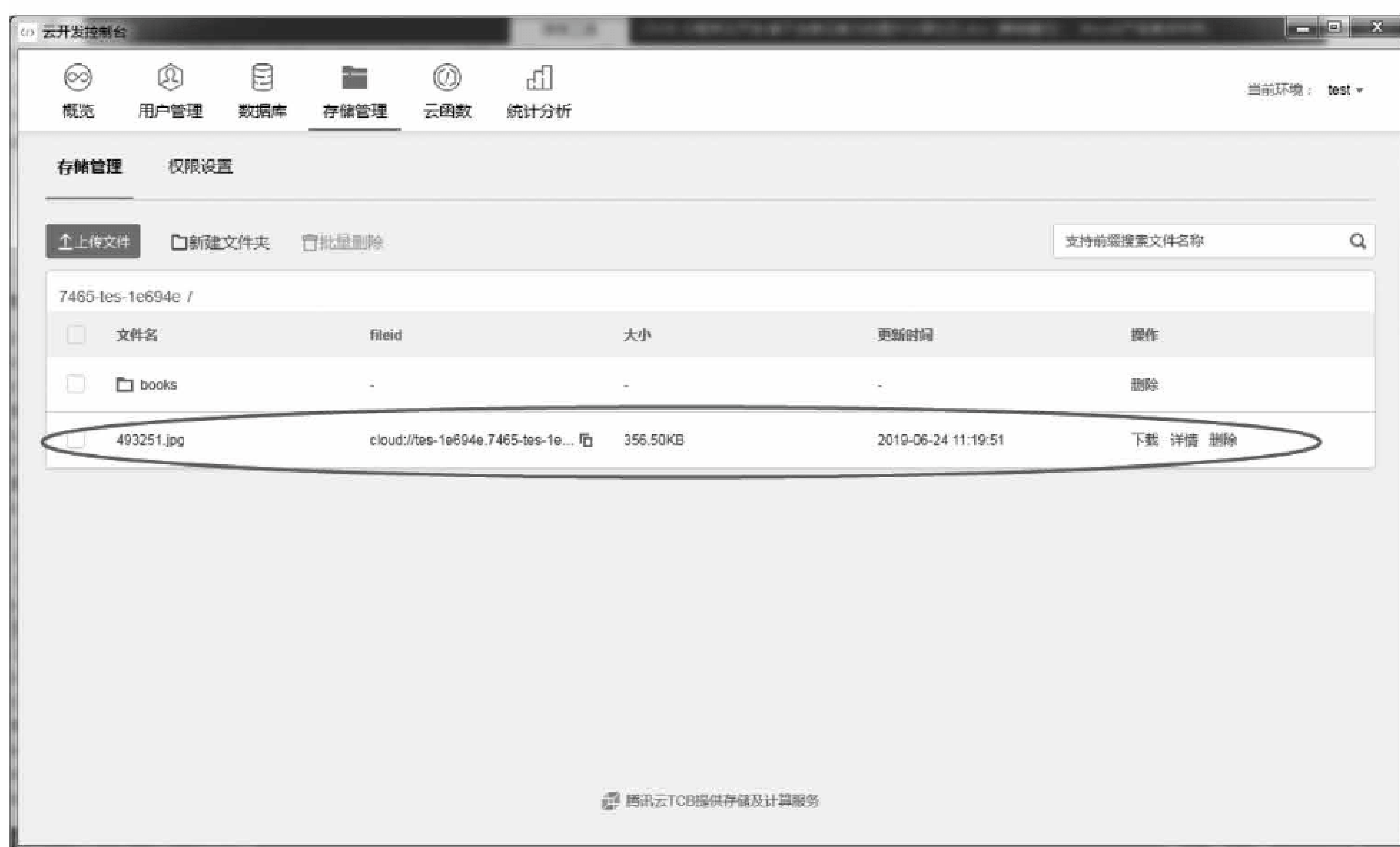
此外还需要注意的是，一定要从首页中点击“上传图片”按钮跳转到当前页面再进行测试，否则可能无法获取到用户的个人信息数据。

点击首页右上角的按钮跳转到当前页面，并从本地任意上传一张图片。

此时云开发控制台的运行效果如图 19-18 所示。



(a) 云数据库的photos集中出现记录



(b) 云存储中出现已上传的图片

图 19-18 云开发控制台显示信息

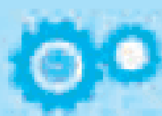
由图可见，本地图片上传成功后会分别出现在云数据库和云存储中。为了后续的图片展示功能，不妨再继续上传若干张本地图片进行测试。



视频讲解

2 展示图片历史记录

在往云数据库中添加记录时，会自动登记当前用户的 openid 并记录成“_openid”字段，因此只要筛选出 openid 属性值与当前用户匹配的所有数据即

可展示图片历史记录。

在 add.js 文件中添加自定义函数 getHistoryPhotos，用于获取当前用户已经上传过的图片数据。相关 JS(pages/add/add.js)代码片段如下：

```
1.  Page({
2.    /**
3.     * 自定义函数 -- 获取已上传图片历史记录
4.     */
5.    getHistoryPhotos: function() {
6.      //获取当前用户的 openid
7.      let openid = app.globalData.openid
8.
9.      //从云数据集查找当前用户的上传记录
10.     photos.where({
11.       _openid:openid
12.     }).get({
13.       success:res =>{
14.         this.setData({historyPhotos:res.data})
15.       }
16.     })
17.   },
18. })
```

上述代码表示从云数据集 photos 中查找当前用户上传的图片记录，并将获取到的记录集存到变量 historyPhotos 中，该名称可以由开发者自定义。

修改 add.wxml 文件中显示图片历史记录的代码，使用 wx:for 循环来展示实际图片。

相关 WXML(pages/add/add.wxml)代码片段如下：

```
1.      <!-- 图片集 -->
2.      <view>
3.        <block wx:for='{{historyPhotos}}' wx:key='history{{index}}'>
4.          <image src='{{item.photoUrl}}'></image>
5.        </block>
6.      </view>
```

最后在 add.js 文件的 onLoad 和 upload 函数中分别引用 getHistoryPhotos，表示打开当前页面和上传新图片后都更新显示图片历史记录。

相关 JS(pages/add/add.js)代码片段如下：

```
1.  Page({
2.    /**
3.     * 自定义函数 -- 上传图片
4.     */
5.    upload: function() {
6.      //选择图片
7.      wx.chooseImage({
8.        count: 1,
9.        sizeType: ['compressed'],
10.       sourceType: ['album', 'camera'],
11.       success: function(res) {
12.         …
13.       },
14.       fail: e => {
15.         console.error(e)
16.       },
```

```
17.         complete: () => {
18.           //上传完成后关闭 loading 提示框
19.           wx.hideLoading()
20.           //更新图片历史记录
21.           this.getHistoryPhotos()
22.         }
23.       })
24.     },
25.
26.     /**
27.      * 生命周期函数--监听页面加载
28.      */
29.     onLoad: function(options) {
30.       //更新图片历史记录
31.       this.getHistoryPhotos()
32.     },
33.
34.   })
```

在已经有上传图片的前提条件下打开上传图片页，此时运行效果如图 19-19 所示。

由图可见，当前页面可以成功显示云存储中的图片，此时上传图片页的逻辑全部完成。



图 19-19　Console 控制台显示用户个人信息

19.3.3　首页逻辑



视频讲解

1 图片列表展示

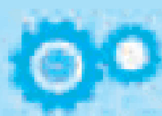
首先在 index.js 文件顶端声明对 photos 数据集的引用，相关 JS(pages/index/index.js)代码片段如下：

```
1.  const db = wx.cloud.database()
2.  const photos = db.collection('photos')
3.
4.  Page({
5.      …
6.  })
```

其中 const photos 是自定义名称，开发者可以自行更改。

在 index.js 文件的 onShow 函数中追加对云端数据库的读取代码，这样从其他页面返回首页时也可以刷新图片列表。相关 JS(pages/index/index.js)代码片段如下：

```
1.  Page({
2.    /**
3.     * 生命周期函数--监听页面显示
4.     */
5.    onShow: function() {
6.      //获取图片列表(按照添加日期降序排列)
7.      photos.orderBy('addDate','desc').get({
8.        success: res => {
9.          this.setData({
10.           photoList: res.data
11.         })
12.       }
```

```
13.         })
14.     },
15.  })
```

修改 index.wxml 文件中卡片的相关代码，使用 wx:for 循环语句显示实际图片信息。相关 WXML(pages/index/index.wxml)代码片段如下：

```
1.     <!-- 卡片 -->
2.     <view class='card' wx:for='{{photoList}}' wx:key='photo{{index}}'>
3.       <!-- 卡片页眉 -->
4.       <view class='card-head'>
5.         <image class='avatar' src='{{item.avatarUrl}}' mode='widthFix'></image>
6.         <view class='title'>
7.           <text class='nickName'>{{item.nickName}}</text>
8.           <text class='address'>{{item.province}},{{item.country}}</text>
9.         </view>
10.      </view>
11.
12.      <!-- 卡片主体 -->
13.      <view class='card-body'>
14.        <image src='{{item.photoUrl}}' mode='widthFix'></image>
15.      </view>
16.
17.      <!-- 卡片页脚 -->
18.      <view class='card-foot'>
19.        <text>{{item.addDate}}</text>
20.      </view>
21.    </view>
22.  </view>
```

为了显示出上传日期的区别，开发者可以临时手动修改云数据集里面 addDate 字段的值，此时运行效果如图 19-20 所示。

由图可见，首页已经可以成功显示当前云存储中的图片和上传者信息，并且按照上传日期将最新的图片显示在最上面。



视频讲解

2 点击头像跳转个人主页

修改 index.wxml 文件中卡片页眉的相关代码，使用<navigator>组件实现头像图片的导航功能。相关 WXML(pages/index/index.wxml)代码片段如下：

```
1.       <!-- 卡片页眉 -->
2.       <view class='card-head'>
3.         <navigator url='../homepage/homepage?id={{item._openid}}'>
4.           <image class='avatar' src='{{item.avatarUrl}}' mode='widthFix'></image>
5.         </navigator>
6.         <view class='title'>
7.           <text class='nickName'>{{item.nickName}}</text>
8.           <text class='address'>{{item.province}},{{item.country}}</text>
9.         </view>
10.      </view>
```

上述代码表示点击头像图片可以跳转到 homepage 页面。由于未来实际使用该小程序时有可能首页会呈现多名不同用户的头像信息，所以在<navigator>组件的 url 属性值中携带自



(a) 页面初始效果



(b) 下拉滚动效果

图 19-20 首页图片列表展示

定义参数 id,用于记录当前被点击用户的 openid。

后续的代码逻辑需要在个人主页(homepage)中实现。



视频讲解

3 点击图片跳转图片展示页

修改 index.wxml 文件中卡片主体的相关代码,使用< navigator >组件实现上传图片的导航功能。相关 WXML(pages/index/index.wxml)代码片段如下:

```
1.     <!-- 卡片主体 -->
2.     <view class = 'card - body'>
3.       <navigator url = '../detail/detail?id = {{item._id}}'>
4.         <image src = '{{item.photoUrl}}' mode = 'widthFix'></image>
5.       </navigator>
6.     </view>
```

上述代码表示点击展示图片可以跳转到 detail 页面。由于首页会呈现不同图片信息,所以在< navigator >组件的 url 属性值中携带自定义参数 id,用于记录当前被点击图片的 id。

后续的代码逻辑需要在图片展示页(detail)中实现。

19.3.4 个人主页逻辑



视频讲解

1 获取被点击用户上传的图片列表

从首页跳转到个人主页时会携带被点击用户的 openid 值,所以可以去云数据集查找该用户上传的所有图片记录。

首先在 homepage.js 文件顶端声明对云数据集 photos 的引用,相关 JS

(pages/homepage/homepage.js)代码片段如下：

```
1.  const db = wx.cloud.database()
2.  const photos = db.collection('photos')
3.
4.  Page({
5.      …
6.  })
```

其中 const photos 是自定义名称，开发者可以自行更改。

然后在 homepage.js 文件的 onLoad 函数中追加对云数据集 photos 的读取代码。

相关 JS(pages/index/index.js)代码片段如下：

```
1.  Page({
2.    /**
3.     * 生命周期函数--监听页面加载
4.     */
5.    onLoad: function(options) {
6.      //获取被点击用户的 openid
7.      let openid = options.id
8.
9.      //显示 loading 提示框
10.     wx.showLoading({
11.       title: '数据加载中',
12.     })
13.
14.     //查找云数据集 photos 中该用户的图片上传记录
15.     photos.orderBy('addDate', 'desc').where({
16.       _openid: openid
17.     }).get({
18.       success: res => {
19.         //获取数据记录
20.         this.setData({
21.           photoList: res.data
22.         })
23.         //关闭 loading 提示框
24.         wx.hideLoading()
25.       }
26.     })
27.   },
28. })
```

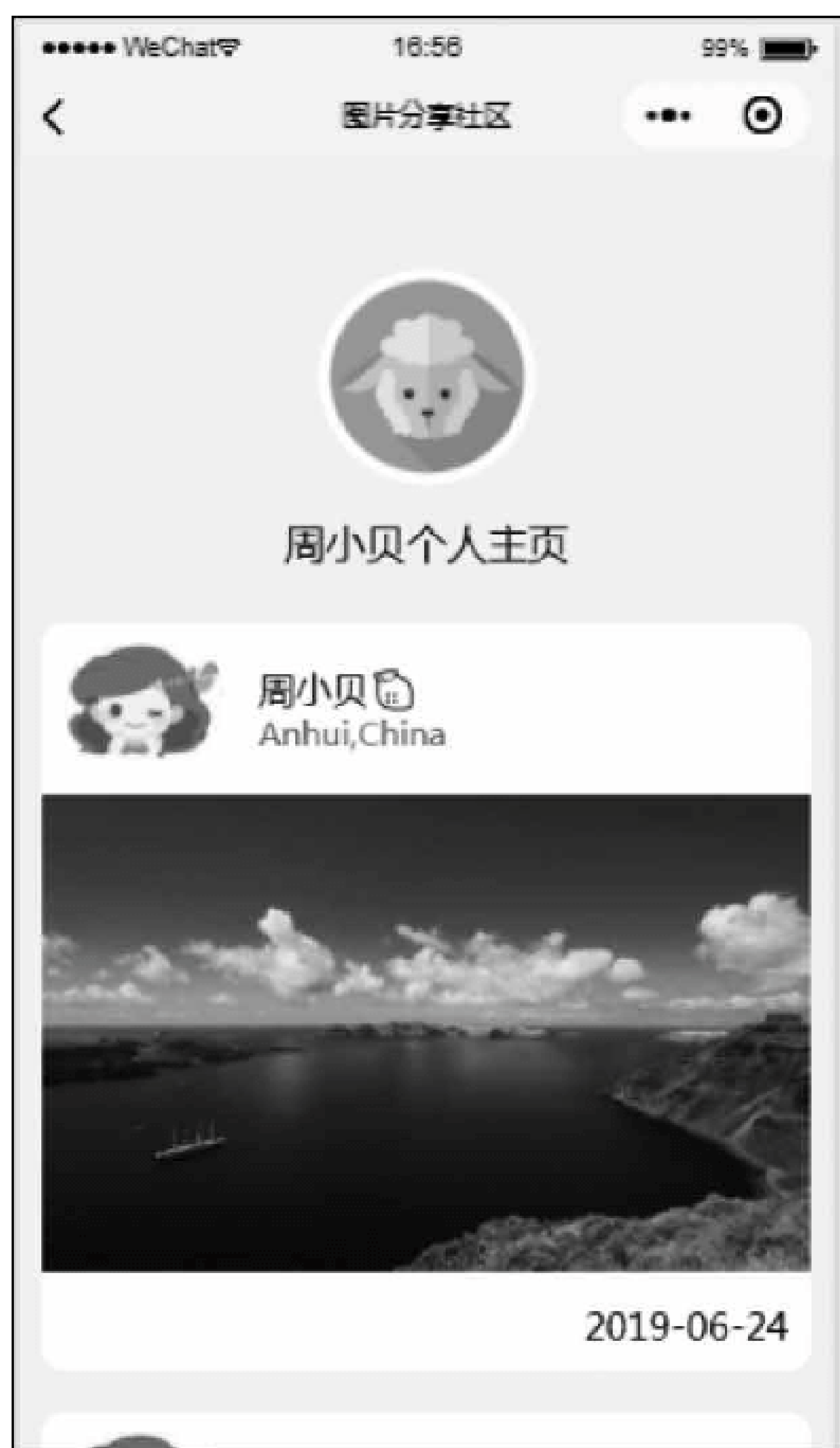
最后修改 homepage.wxml 中卡片的代码，使用 wx:for 循环语句显示实际图片信息。

相关 WXML(pages/homepage/homepage.wxml)代码片段如下：

```
1.    <!-- 卡片 -->
2.    <view class = 'card' wx:for = '{{photoList}}' wx:key = 'photo{{index}}'>
3.      <!-- 卡片页眉 -->
4.      <view class = 'card-head'>
5.        <image class = 'avatar' src = '{{item.avatarUrl}}' mode = 'widthFix'></image>
6.        <view class = 'title'>
7.          <text class = 'nickName'>{{item.nickName}}</text>
8.          <text class = 'address'>{{item.province}},{{item.country}}</text>
9.        </view>
10.     </view>
11.
12.     <!-- 卡片主体 -->
```

```
13.         <view class='card-body'>
14.           <image src='{{item.photoUrl}}' mode='widthFix'></image>
15.         </view>
16.
17.         <!-- 卡片页脚 -->
18.         <view class='card-foot'>
19.           <text>{{item.addDate}}</text>
20.         </view>
21.       </view>
```

此时运行效果如图 19-21 所示。



(a) 页面初始效果



(b) 下拉滚动效果

图 19-21 个人主页的图片列表展示

由图可见，个人主页已经可以成功显示云存储中的图片，并且按照上传日期将最新的图片显示在最上面。

由于尚未实现顶端用户头像和昵称更新的逻辑，所以目前仍然是测试图片效果。

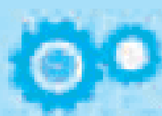


视频讲解

2 显示顶端用户头像和昵称

修改 homepage.wxml 文件中顶端头像和昵称的代码，使用 photoList 数据中的任意一条均可获取头像和昵称属性值。相关 WXML（pages/homepage/homepage.wxml）代码片段如下：

```
1.    <!-- 头像和昵称 -->
2.    <view class='avatarBox'>
3.      <image src='{{photoList[0].avatarUrl}}'></image>
4.      <text>{{photoList[0].nickName}}个人主页</text>
5.    </view>
```



此时运行效果如图 19-22 所示。

由图可见,此时已经可以成功显示当前用户的个人信息了。

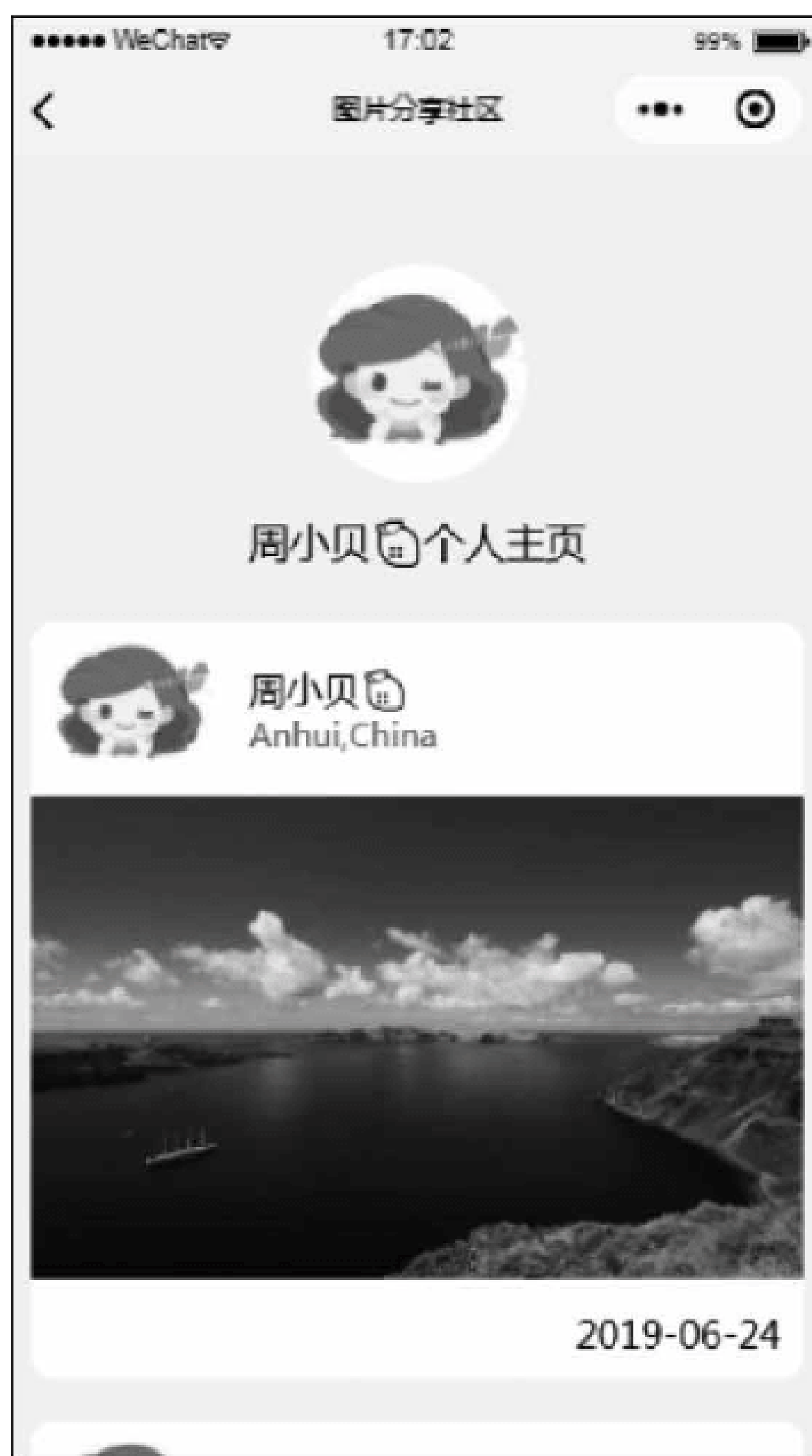


图 19-22 个人主页的顶端头像和昵称展示

3 点击图片跳转图片展示页


视频讲解

修改 homepage.wxml 文件中卡片主体的相关代码,使用< navigator >组件实现上传图片的导航功能。相关 WXML(pages/homepage/homepage.wxml)代码片段如下:

```
1.      <!-- 卡片主体 -->
2.      <view class = 'card - body'>
3.        <navigator url = '../detail/detail? id = {{item._id}}'>
4.          <image src = '{{item.photoUrl}}' mode = 'widthFix'></image>
5.        </navigator>
6.      </view>
```

上述代码表示点击展示图片可以跳转到 detail 页面。由于首页会呈现不同图片信息,所以在< navigator >组件的 url 属性值中携带自定义参数 id,用于记录当前被点击图片的 id。

后续的代码逻辑需要在图片展示页(detail)中实现。

19.3.5 图片展示页逻辑


视频讲解

1 显示当前图片

从首页或个人主页的图片卡片跳转到图片展示页时均会携带被点击图片的 id,所以可以去云数据集查找该图片的记录。

首先在 detail.js 文件顶端声明对云数据集 photos 的引用,相关 JS(pages/detail/detail.js)代码片段如下:

```
1.  const db = wx.cloud.database()
2.  const photos = db.collection('photos')
3.
4.  Page({
5.      …
6.  })
```

其中 const photos 是自定义名称,开发者可以自行更改。

然后在 detail.js 文件的 onLoad 函数中追加对云数据集 photos 的读取代码。

相关 JS(pages/detail/detail.js)代码片段如下:

```
1.  Page({
2.    /**
3.     * 生命周期函数--监听页面加载
4.     */
5.    onLoad: function(options) {
6.      //根据图片 id 获取云数据集中的图片记录
7.      photos.doc(options.id).get({
8.        success: res => {
```

```
9.            this.setData({ photo: res.data })
10.         }
11.       })
12.     },
13.  })
```

最后修改 detail.wxml 中图片的代码，显示实际图片信息。

相关 WXML(pages/detail/detail.wxml)代码片段如下：

```
1.  <!-- 顶端图片展示 -->
2.  <image src='{{photo.photoUrl}}' mode='widthFix'></image>
```

从首页点击任意图片，此时运行效果如图 19-23 所示。



(a) 首页效果



(b) 图片展示页效果

图 19-23 图片展示页显示顶端图片

由图可见，此时点击首页卡片列表中的任意图片均可在图片展示页显示正确的画面。实际上从个人主页点击图片进入图片展示页也是一样的效果，这里不再另行演示。

2 下载图片到本地设备



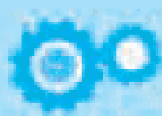
视频讲解

修改 detail.wxml 中相关按钮的代码，为其添加点击事件 downloadPhoto。

相关 WXML(pages/detail/detail.wxml)代码片段如下：

```
1.  <!-- 按钮区域 -->
2.  <view>
3.    <button type='primary' plain bindtap='downloadPhoto'>下载到本地</
button>
4.    <button type='primary' plain>分享给好友</button>
5.    <button type='primary' plain>全屏预览</button>
6.  </view>
```

然后在 detail.js 文件中追加自定义函数 downloadPhoto，相关 JS(pages/detail/detail.js)



代码片段如下：

```
1.  Page({
2.    /**
3.     * 自定义函数 -- 下载图片到本地
4.     */
5.    downloadPhoto: function() {
6.      //从云存储中进行图片下载
7.      wx.cloud.downloadFile({
8.        fileID: this.data.photo.photoUrl,
9.        success: res => {
10.         //保存图片到本地相册
11.         wx.saveImageToPhotosAlbum({
12.           filePath: res.tempFilePath,
13.           success: res => {
14.             wx.showToast({
15.               title: '保存成功!',
16.             })
17.           },
18.           fail: err => {
19.             wx.showToast({
20.               title: '保存失败!',
21.               icon: 'none'
22.             })
23.           }
24.         })
25.       },
26.       fail: err => {
27.         console.log(err)
28.       }
29.     })
30.   },
31. })
```

此时运行效果如图 19-24 所示。



(a) 点击按钮进行图片下载



(b) 保存成功提示

图 19-24　下载图片到本地设备效果

当前是计算机端模拟运行效果，可以自选图片另存为地址；如果用手机测试会将图片直接保存到手机相册中。

3 分享图片给好友

视频讲解

修改 detail.wxml 中相关按钮的代码，为其添加 open-type 属性。

相关 WXML(pages/detail/detail.wxml)代码片段如下：

```
1.  <!-- 按钮区域 -->
2.  <view>
3.    <button type='primary' plain bindtap='downloadPhoto'>下载到本地</button>
4.    <button type='primary' plain open-type='share'>分享给好友</button>
5.    <button type='primary' plain>全屏预览</button>
6.  </view>
```

此时点击“分享给好友”按钮的效果就等同于点击右上角的“…”，都可以进行页面分享。

如果用户需要自定义分享的内容，可以在 detail.js 文件中修改自带的 onShareAppMessage 函数，相关 JS(pages/detail/detail.js)代码片段如下：

```
1.  Page({
2.    /**
3.     * 用户点击右上角分享
4.     */
5.    onShareAppMessage: function() {
6.      return {
7.        title: '给你分享一张好看的图片',
8.        path: 'pages/detail/detail?id=' + this.data.photo._id
9.      }
10.   }
11. })
```

此时运行效果如图 19-25 所示。

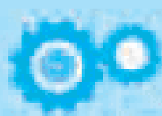


(a) 点击按钮进行图片分享



(b) 图片分享对话框

图 19-25　分享图片效果



当前是计算机端模拟运行效果，如果用手机测试还可以选择微信好友列表进行发送。

4 全屏预览图片


视频讲解

修改 detail.wxml 中相关按钮的代码，为其添加点击事件 previewPhoto。

相关 WXML(pages/detail/detail.wxml)代码片段如下：

```
1.  <!-- 按钮区域 -->
2.  <view>
3.    <button type = 'primary' plain bindtap = 'downloadPhoto'>下载到本地</
button>
4.    <button type = 'primary' plain open - type = 'share'>分享给好友</button>
5.      <button type = 'primary' plain bindtap = 'previewPhoto'>全屏预览</button>
6.  </view>
```

然后在 detail.js 文件中追加自定义函数 previewPhoto，相关 JS(pages/detail/detail.js)代码片段如下：

```
1.  Page({
2.    /**
3.     * 自定义函数--全屏预览图片
4.     */
5.    previewPhoto: function() {
6.      //定义图片 URL 地址
7.      var urls = [this.data.photo.photoUrl]
8.      //全屏预览图片
9.      wx.previewImage({
10.       urls: urls,
11.     })
12.   },
13. })
```

此时运行效果如图 19-26 所示。



(a) 点击按钮进行图片全屏预览



(b) 全屏预览效果

图 19-26　全屏预览图片效果

当前是计算机端模拟运行效果，如果用手机预览，长按还可以保存图片到手机相册中。



19.4 完整代码展示



19.4.1 应用文件代码展示

app.json 文件的完整代码如下：

```
1.  {
2.    "pages": [
3.      "pages/index/index",
4.      "pages/homepage/homepage",
5.      "pages/detail/detail",
6.      "pages/add/add"
7.    ],
8.    "window": {
9.      "navigationBarBackgroundColor": "#F6F6F6",
10.     "navigationBarTitleText": "图片分享社区",
11.     "navigationBarTextStyle": "black"
12.   }
13. }
```

app.wxss 文件的完整代码如下：

```
1.  /*1.整体容器样式*/
2.  .container {
3.    display: flex;
4.    flex-direction: column;
5.    align-items: center;
6.    background-color: whitesmoke;
7.    min-height: 100vh;
8.  }
9.
10. /*2.卡片区域整体样式*/
11. .card {
12.   width: 710rpx;
13.   border-radius: 20rpx;
14.   background-color: white;
15.   margin: 20rpx 0;
16. }
17. /*2-1 卡片页眉*/
18. .card-head {
19.   display: flex;
20.   flex-direction: row;
21.   align-items: center;
22. }
23.
24. /*2-1-1 头像图片*/
25. .avatar {
26.   width: 150rpx;
27.   border-radius: 50%;
28.   margin: 0 20rpx;
29. }
30.
```

```
31. /* 2 - 1 - 2 文字信息 */
32. .title {
33.   display: flex;
34.   flex - direction: column;
35. }
36. /* 2 - 1 - 2(1) 昵称 */
37. .nickName {
38.   font - size: 35rpx;
39.   margin: 0 10rpx;
40. }
41. /* 2 - 1 - 2(2) 所在地区 */
42. .address {
43.   font - size: 30rpx;
44.   color: gray;
45.   margin: 0 10rpx;
46. }
47. /* 2 - 2 卡片主体 */
48. .card - body image {
49.   width: 100 % ;
50. }
51. /* 2 - 3 卡片页脚 */
52. .card - foot{
53.   font - size: 35rpx;
54.   text - align: right;
55.   margin: 20rpx;
56. }
```

app.js 文件的完整代码如下(该文件代码为云模板自动生成)：

```
1.  App({
2.    onLaunch: function() {
3.
4.      if (!wx.cloud) {
5.        console.error('请使用 2.2.3 或以上的基础库以使用云能力')
6.      } else {
7.        wx.cloud.init({
8.          traceUser: true,
9.        })
10.     }
11.
12.     this.globalData = {}
13.   }
14. })
```

19.4.2 云函数文件代码展示

云函数 getOpenid 的 JS 文件(getOpenid/index.js)的完整代码如下：

```
1.  //云函数入口文件
2.  const cloud = require('wx - server - sdk')
3.
4.  cloud.init()
5.
6.  //云函数入口函数
7.  exports.main = async (event, context) => {
```

```
8.     const wxContext = cloud.getWXContext()
9.
10.    return {
11.      openid: wxContext.OPENID
12.    }
13.  }
```

19.4.3 页面文件代码展示

1 首页代码展示

WXML 文件(pages/index/index.wxml)的完整代码如下：

```
1.  <view class='container'>
2.    <!-- 卡片 -->
3.    <view class='card' wx:for='{{photoList}}' wx:key='photo{{index}}'>
4.      <!-- 卡片页眉 -->
5.      <view class='card-head'>
6.        <navigator url='../homepage/homepage?id={{item._openid}}'>
7.          <image class='avatar' src='{{item.avatarUrl}}' mode='widthFix'></image>
8.        </navigator>
9.        <view class='title'>
10.         <text class='nickName'>{{item.nickName}}</text>
11.         <text class='address'>{{item.province}},{{item.country}}</text>
12.       </view>
13.     </view>
14.
15.     <!-- 卡片主体 -->
16.     <view class='card-body'>
17.       <navigator url='../detail/detail?id={{item._id}}'>
18.         <image src='{{item.photoUrl}}' mode='widthFix'></image>
19.       </navigator>
20.     </view>
21.
22.     <!-- 卡片页脚 -->
23.     <view class='card-foot'>
24.       <text>{{item.addDate}}</text>
25.     </view>
26.   </view>
27. </view>
28.
29. <!-- "上传图片"按钮 -->
30. <view class='floatBtn'>
31.   <button size='mini' type='primary' bindtap='goToAdd' open-type='getUserInfo' bindgetuserinfo='getUserInfo'>上传图片</button>
32. </view>
```

WXSS 文件(pages/index/index.wxss)的完整代码如下：

```
1.  /* 浮动按钮样式 */
2.  .floatBtn{
3.    position: fixed;
4.    top: 50rpx;
5.    right: 50rpx;
6.  }
```

JS 文件(pages/index/index.js)的完整代码如下：

```
1.  const db = wx.cloud.database()
2.  const photos = db.collection('photos')
3.  var app = getApp()
4.
5.  Page({
6.    /**
7.     * 自定义函数--跳转上传图片页
8.     */
9.    goToAdd: function(options) {
10.     wx.navigateTo({
11.       url: '../add/add',
12.     })
13.   },
14.
15.   /**
16.    * 自定义函数--获取用户个人信息
17.    */
18.   getUserInfo: function(e) {
19.     //尝试打印输出个人信息,测试是否获取成功
20.     //console.log(e.detail.userInfo)
21.
22.     //将用户个人信息存放到全局变量 userInfo 中
23.     app.globalData.userInfo = e.detail.userInfo
24.
25.     //检测是否已经获取过了用户 openid 信息
26.     if (app.globalData.openid == null) {
27.       //如果是第一次登录,则使用云函数获取用户 openid
28.       wx.cloud.callFunction({
29.         name: 'getOpenid',
30.         complete: res => {
31.           //尝试打印输出个人信息,测试是否获取成功
32.           //console.log(res.result.openid)
33.           //将用户 openid 信息存放到全局变量 openid 中
34.           app.globalData.openid = res.result.openid
35.         }
36.       })
37.     }
38.   },
39.   /**
40.    * 生命周期函数--监听页面显示
41.    */
42.   onShow: function() {
43.     //获取图片列表(按照添加日期降序排列)
44.     photos.orderBy('addDate','desc').get({
45.       success: res => {
46.         this.setData({
47.           photoList: res.data
48.         })
49.       }
50.     })
51.   }
52. })
```

2 个人主页代码展示

WXML 文件(pages/homepage/homepage.wxml)的完整代码如下:

```
1.  <view class = 'container'>
2.    <!-- 头像和昵称 -->
3.    <view class = 'avatarBox'>
4.      <image src = '{{photoList[0].avatarUrl}}'></image>
5.      <text>{{photoList[0].nickName}}个人主页</text>
6.    </view>
7.
8.    <!-- 卡片 -->
9.    <view class = 'card' wx:for = '{{photoList}}' wx:key = 'photo{{index}}'>
10.     <!-- 卡片页眉 -->
11.     <view class = 'card-head'>
12.       <navigator url = '../user/user?id = {{item._openid}}'>
13.         <image class = 'avatar' src = '{{item.avatarUrl}}' mode = 'widthFix'></image>
14.       </navigator>
15.       <view class = 'title'>
16.         <text class = 'nickName'>{{item.nickName}}</text>
17.         <text class = 'address'>{{item.province}},{{item.country}}</text>
18.       </view>
19.     </view>
20.
21.     <!-- 卡片主体 -->
22.     <view class = 'card-body'>
23.       <navigator url = '../detail/detail?id = {{item._id}}'>
24.         <image src = '{{item.photoUrl}}' mode = 'widthFix'></image>
25.       </navigator>
26.     </view>
27.
28.     <!-- 卡片页脚 -->
29.     <view class = 'card-foot'>
30.       <text>{{item.addDate}}</text>
31.     </view>
32.   </view>
33. </view>
```

WXSS 文件(pages/homepage/homepage.wxss)的完整代码如下:

```
1.  /* 头像区域 */
2.  .avatarBox{
3.    display: flex;
4.    flex-direction: column;
5.    align-items: center;
6.    padding-top: 100rpx;
7.  }
8.  /* 头像区域中的头像图片 */
9.  .avatarBox image{
10.   width: 200rpx;
11.   height: 200rpx;
12.   border-radius: 50%;
13. }
14. /* 头像区域中的昵称文本 */
15. .avatarBox text{
16.   margin: 30rpx 0;
17. }
```

JS 文件(pages/homepage/homepage.js)的完整代码如下：

```
1.  const db = wx.cloud.database()
2.  const photos = db.collection('photos')
3.
4.  Page({
5.    /**
6.     * 生命周期函数--监听页面加载
7.     */
8.    onLoad: function(options) {
9.      //获取被点击用户的 openid
10.     let openid = options.id
11.
12.     //显示 loading 提示框
13.     wx.showLoading({
14.       title: '数据加载中',
15.     })
16.
17.     //查找云数据集 photos 中该用户的图片上传记录
18.     photos.orderBy('addDate', 'desc').where({
19.       _openid: openid
20.     }).get({
21.       success: res => {
22.         //获取数据记录
23.         this.setData({
24.           photoList: res.data
25.         })
26.         //关闭 loading 提示框
27.         wx.hideLoading()
28.       }
29.     })
30.   }
31. })
```

3 图片展示页代码展示

WXML 文件(pages/detail/detail.wxml)的完整代码如下：

```
1.  <!-- 顶端图片展示 -->
2.  <image src='{{photo.photoUrl}}' mode='widthFix'></image>
3.
4.  <!-- 按钮区域 -->
5.  <view>
6.    <button type='primary' plain bindtap='downloadPhoto'>下载到本地</button>
7.    <button type='primary' plain open-type='share'>分享给好友</button>
8.      <button type='primary' plain bindtap='previewPhoto'>全屏预览</button>
9.  </view>
```

WXSS 文件(pages/detail/detail.wxss)的完整代码如下：

```
1.  /* 顶端展示图片样式 */
2.  image{
3.    width: 750rpx;
4.  }
5.  /* 按钮外层 view */
6.  view{
7.    padding: 0 50rpx;
```

```
8.  }
9.  /* 按钮样式 */
10. button{
11.   margin: 10rpx;
12. }
```

JS 文件(pages/detail/detail.js)的完整代码如下：

```
1.  const db = wx.cloud.database()
2.  const photos = db.collection('photos')
3.
4.  Page({
5.    /**
6.     * 自定义函数 -- 下载图片到本地
7.     */
8.    downloadPhoto: function() {
9.      //从云存储中进行图片下载
10.     wx.cloud.downloadFile({
11.       fileID: this.data.photo.photoUrl,
12.       success: res => {
13.         //保存图片到本地相册
14.         wx.saveImageToPhotosAlbum({
15.           filePath: res.tempFilePath,
16.           success: res => {
17.             wx.showToast({
18.               title: '保存成功!',
19.             })
20.           },
21.           fail: err => {
22.             wx.showToast({
23.               title: '保存失败!',
24.               icon: 'none'
25.             })
26.           }
27.         })
28.       },
29.       fail: err => {
30.         console.log(err)
31.       }
32.     })
33.   },
34.   /**
35.    * 自定义函数 -- 全屏预览图片
36.    */
37.   previewPhoto: function() {
38.     //定义图片 URL 地址
39.     var urls = [this.data.photo.photoUrl]
40.     //全屏预览图片
41.     wx.previewImage({
42.       urls: urls,
43.     })
44.   },
45.   /**
46.    * 生命周期函数 -- 监听页面加载
47.    */
48.   onLoad: function(options) {
```

```
49.      //根据图片 id 获取云数据集中的图片记录
50.      photos.doc(options.id).get({
51.        success: res => {
52.          this.setData({ photo: res.data })
53.        }
54.      })
55.    },
56.    /**
57.     * 用户点击右上角分享
58.     */
59.    onShareAppMessage: function() {
60.      return {
61.        title: '给你分享一张好看的图片',
62.        path: 'pages/detail/detail?id=' + this.data.photo._id
63.      }
64.    }
65.  })
```

4 上传图片页代码展示

WXML 文件(pages/add/add.wxml)的完整代码如下：

```
1.  <view class='container'>
2.    <!-- "点击此处上传图片"按钮 -->
3.    <button type='warn' bindtap='upload'>点击此处上传图片</button>
4.
5.    <!-- 已上传图片区域 -->
6.    <view class='photoBox'>
7.      <!-- 标题 -->
8.      <text>已上传图片历史记录</text>
9.
10.     <!-- 图片集 -->
11.     <view>
12.       <block wx:for='{{historyPhotos}}' wx:key='history{{index}}'>
13.         <image src='{{item.photoUrl}}'></image>
14.       </block>
15.     </view>
16.   </view>
17. </view>
```

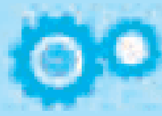
WXSS 文件(pages/add/add.wxss)的完整代码如下：

```
1.  /*已上传图片区域*/
2.  .photoBox{
3.    margin-top: 50rpx;
4.    display: flex;
5.    flex-direction: column;
6.  }
7.  /*标题样式*/
8.  .photoBox text{
9.    margin-bottom: 30rpx;
10.   text-align: center;
11. }
12. /*图片样式*/
13. .photoBox image{
14.   float: left;
15.   width: 250rpx;
```

```
16.     height: 250rpx;
17.  }
```

JS 文件(pages/add/add.js)的完整代码如下：

```
1.  const db = wx.cloud.database()
2.  const photos = db.collection('photos')
3.  var app = getApp()
4.
5.  //格式化当前日期
6.  function formatDate() {
7.    var now = new Date()
8.    var year = now.getFullYear()
9.    var month = now.getMonth() + 1
10.   var day = now.getDate()
11.
12.   if (month < 10) month = '0' + month
13.   if (day < 10) day = '0' + day
14.
15.   return year + '-' + month + '-' + day
16. }
17.
18. Page({
19.   /**
20.    * 自定义函数 -- 上传图片
21.    */
22.   upload: function() {
23.     //选择图片
24.     wx.chooseImage({
25.       count: 1,
26.       sizeType: ['compressed'],
27.       sourceType: ['album', 'camera'],
28.       success: function(res) {
29.         //loading 提示框表示正在上传图片
30.         wx.showLoading({
31.           title: '上传中',
32.         })
33.         //获取图片临时地址
34.         const filePath = res.tempFilePaths[0]
35.
36.         //自定义云端的图片名称
37.         const cloudPath = Math.floor(Math.random() * 1000000) + filePath.match(/\.[^.]
+?$/)[0]
38.         //上传图片到云存储空间中
39.         wx.cloud.uploadFile({
40.           cloudPath,
41.           filePath,
42.           success: res => {
43.             //提示上传成功
44.             wx.showToast({
45.               title: '上传成功!',
46.               duration: 3000
47.             })
48.
49.             //获取用户个人基础信息
50.             let userInfo = app.globalData.userInfo
```



```
51.              //获取当天日期
52.              let today = formatDate()
53.
54.              //往云数据集中添加一条记录
55.              photos.add({
56.                data: {
57.                  photoUrl: res.fileID,
58.                  avatarUrl: userInfo.avatarUrl,
59.                  country: userInfo.country,
60.                  province: userInfo.province,
61.                  nickName: userInfo.nickName,
62.                  addDate: today
63.                },
64.                success: res => {
65.                  console.log(res)
66.                },
67.                fail: e => {
68.                  console.log(e)
69.                }
70.              })
71.            },
72.            fail: e => {
73.              //提示上传失败
74.              wx.showToast({
75.                icon: 'none',
76.                title: '上传失败',
77.              })
78.            }
79.          })
80.        },
81.        fail: e => {
82.          console.error(e)
83.        },
84.        complete: () => {
85.          //上传完成后关闭 loading 提示框
86.          wx.hideLoading()
87.          //更新图片历史记录
88.          this.getHistoryPhotos()
89.        }
90.      })
91.    },
92.    /**
93.     * 自定义函数--获取已上传图片历史记录
94.     */
95.    getHistoryPhotos: function() {
96.      //获取当前用户的 openid
97.      let openid = app.globalData.openid
98.
99.      //从云数据集查找当前用户的上传记录
100.     photos.where({
101.       _openid: openid
102.     }).get({
103.       success: res => {
104.         this.setData({
105.           historyPhotos: res.data
```

```
106.          })
107.        }
108.      })
109.    },
110.    /**
111.      * 生命周期函数--监听页面加载
112.      */
113.    onLoad: function(options) {
114.      //尝试输出用户基础信息
115.      //console.log(app.globalData.userInfo)
116.      //尝试输出用户 openid
117.      //console.log(app.globalData.openid)
118.
119.      //更新图片历史记录
120.      this.getHistoryPhotos()
121.    }
122.})
```



第20章 Chapter 20

小程序UI组件库·基于Vant Weapp的生日管家

为了提高小程序的开发效率，可以考虑使用第三方 UI 组件来实现界面的视觉统一。开发者可以方便地引用已经事先设计好的自定义组件来快速搭建小程序页面。本章以有赞第三方 UI 组件库 Vant Weapp 为例，介绍了如何使用自定义组件配合云开发中的数据库基本功能实现一个生日管家小程序。

本章学习目标

- 了解小程序自定义组件的概念；
- 掌握 UI 组件库 Vant Weapp 的下载和使用；
- 掌握小程序云开发中小程序端数据库的相关 API。



20.1 小程序自定义组件简介



20.1.1 什么是自定义组件

从基础库版本 1.6.3 开始，小程序开始支持简洁的组件化编程。开发者可以将页面内的功能模块制作成自定义组件，以便在不同的页面中重复使用；也可以将复杂内容拆分成若干个低耦合的模块，这样有助于代码的后期维护。

例如，目前小程序的原生组件中是没有卡片组件的，开发者可以自行使用图片、按钮、文本等内容通过样式布局组合成一个商品展示卡片，然后自由应用于多个页面上，这种组件就是自定义组件。

20.1.2 自定义组件的引用方式

如果需要开发者自定义组件，则需要为每个组件编写一套由 WXML、WXSS、JSON 以及 JS 几个文件组成的模板代码，并且使用前需要在对应页面的 JSON 文件中进行引用声明。其语法格式如下：

```
{
  "usingComponents": {
    "component-tag-name": "path/to/the/custom/component"
  }
}
```

其中"component-tag-name"换成自定义的组件名称(也就是未来在页面上引用的组件标签名)、"path/to/the/custom/component"换成自定义组件所在的路径地址即可使用。

在完成引用声明后,自定义组件在使用时与小程序原生的基础组件用法非常相似。

20.1.3 小程序 UI 组件库 Vant Weapp

目前市面上提供了一些免费开源的第三方小程序 UI 组件库,可以下载后放到项目文件夹中直接使用,比起开发者从头开始自定义组件更为方便、高效。

这里以有赞小程序组件库 Vant Weapp 为例,从零开始讲解如何使用其开发完成一款生日管家小程序项目。Vant Weapp 是一款轻量、可靠的小程序 UI 组件库,与有赞移动端组件库 Vant 基于相同的视觉规范,并提供一致的 API 接口,方便开发者快速搭建小程序应用。

例如其中的一款卡片组件< van-card >就可以实现如图 20-1 所示的商品展示效果。



图 20-1 Vant Weapp 卡片组件的展示效果

其官方文档地址为"https://youzan.github.io/vant-weapp/#/intro",开发者可以通过查看文档了解这些第三方组件的引用方式和用法示例。

20.1.4 Vant Weapp 的下载和安装

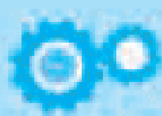
Vant Weapp 有 3 种安装方式,开发者可以根据自己的实际情况任选一种。

1 方法一:通过 npm 安装

小程序目前已经支持使用 npm 安装第三方包,语法如下:

```
# npm
npm i vant-weapp -S --production

# yarn
yarn add vant-weapp --production
```



这种方法需要开发者对 npm 有一定的了解。

2 方法二：通过 git 下载源代码

开发者也可以通过 git 下载源代码，并手动将其中的 dist 或 lib 文件夹复制到项目内的自定义路径地址中，语法如下：

```
git clone https://github.com/youzan/vant-weapp.git
```

这种方法需要开发者对 github 有一定的了解，最好有自己的账号。

3 方法三：直接下载源代码

如果开发者不太熟悉以上两种方法，还可以直接访问网页下载源代码，解压缩后手动将目录中的 dist 或 lib 复制到项目内的自定义路径地址中，其官方的下载访问地址是“https://github.com/youzan/vant-weapp”，然后单击右侧的 Clone or download 按钮，如图 20-2 所示，在下拉列表中单击 Download ZIP 下载压缩包。

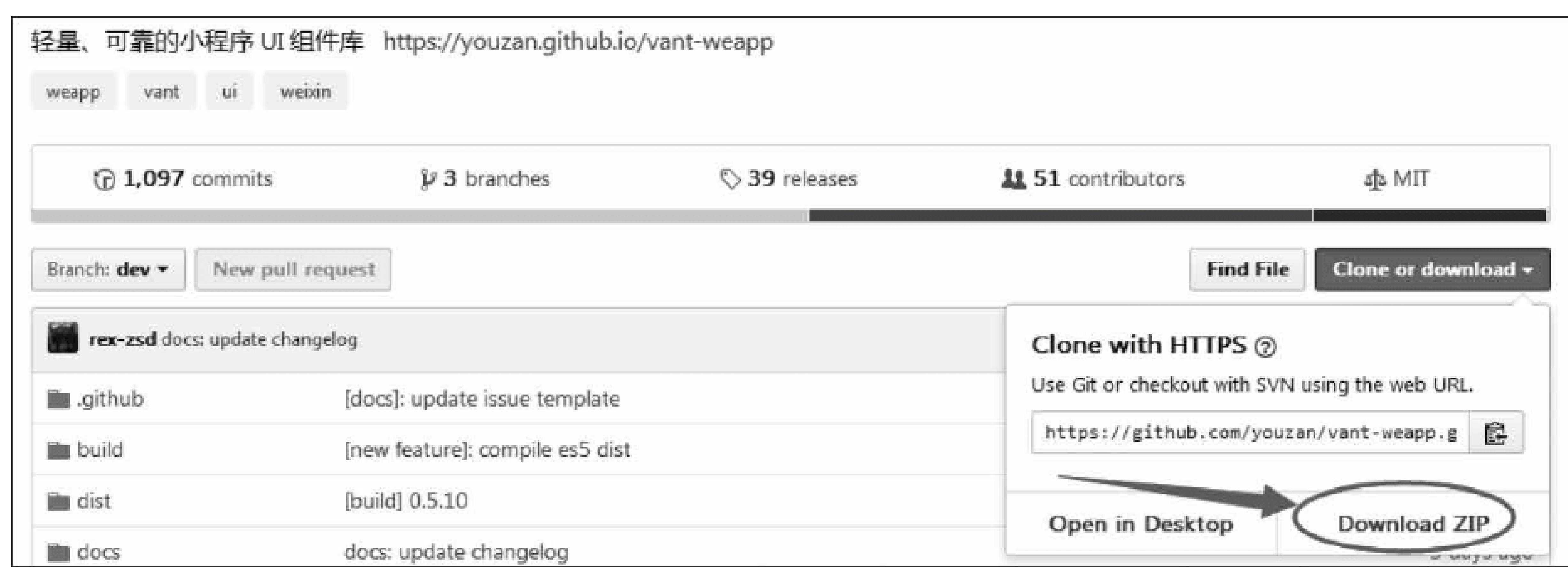
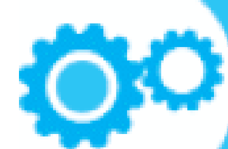


图 20-2 在线下载源代码示意图



20.2 需求分析



本项目一共需要 3 个页面，即首页、好友信息编辑页和好友信息展示页。



视频讲解

20.2.1 首页功能需求

首页功能需求如下：

(1) 首页需要包含“添加新朋友”按钮、搜索框和好友列表。

(2) 点击“添加新朋友”按钮后跳转到好友信息编辑页，可以录入数据。

(3) 搜索框可以根据好友姓名关键词查找指定好友。

(4) 好友生日列表需要展示好友姓名、生日、距离下个生日还有多少天，并且点击好友头像可以跳转到好友信息展示页查看详情。

20.2.2 好友信息编辑页功能需求

好友信息编辑页功能需求如下：

(1) 以表单的形式要求用户录入好友的姓名、性别、生日、电话、关系等信息。

(2) “保存记录”按钮用于添加或更新好友数据到云数据库中。

(3) “取消修改”按钮用于取消本次填写，返回上一页。

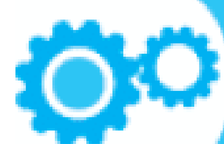
20.2.3 好友信息展示页功能需求

好友信息展示页功能需求如下：

(1) 顶端展示好友的头像和姓名。

(2) 中间列表展示好友的性别、电话、和用户本人的关系、距离出生已经多少天、距离下个生日还有多少天。

(3) 下方是“修改”按钮和“删除”按钮，分别用于跳转编辑页面和直接删除当前好友。



20.3 初始化项目



20.3.1 创建云模板项目



视频讲解

首先需要创建一个云开发项目，在任意盘符下创建一个空白文件夹(例如birthdayMemo)，然后填入AppID和选中“小程序·云开发”，如图20-3所示。

接着删除其中无用的模板代码：



图 20-3 云模板项目填写效果示意图

(1) 删除 cloudFunctions 文件夹下的默认云函数 login 的全部内容。

(2) 找到 app.json 文件，打开并删除其中的页面引用，只保留第一个 pages/index/index。

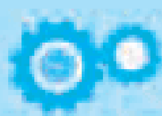
(3) 打开硬盘中的 pages 文件夹，删除 index 以外的所有目录。

(4) 进入 index 文件夹，删除多余的图片，以及 JS、WXML 和 WXSS 文件中的全部代码(JS 文件删除干净后输入关键词 page 让它自动生成 Page({})函数的相关代码)。

(5) 删除 app.wxss 中的代码。

(6) 删除 image 文件夹中的所有图片。

(7) 删除 style 文件夹，此时项目清理全部完成。



20.3.2 部署云数据库



视频讲解

具体操作如下：

(1) 打开云开发控制台，创建一个新的数据集，例如 birthday。

(2) 检查 birthday 数据集的权限，确认是“仅创建者及管理员可读写”，完成。

效果如图 20-4 所示。



图 20-4 在云开发控制台添加数据集 birthday 并设置权限

20.3.3 创建页面文件



视频讲解

本项目有 3 个页面文件，分别是 index(首页)、edit(好友信息编辑页)和 detail(好友信息展示页)。打开 app.json 文件，在 pages 属性中追加 edit 和 detail 页面的路径描述，如图 20-5 所示。

保存后会自动生成后面两个页面，此时 pages 文件夹内部的目录结构如图 20-6 所示。

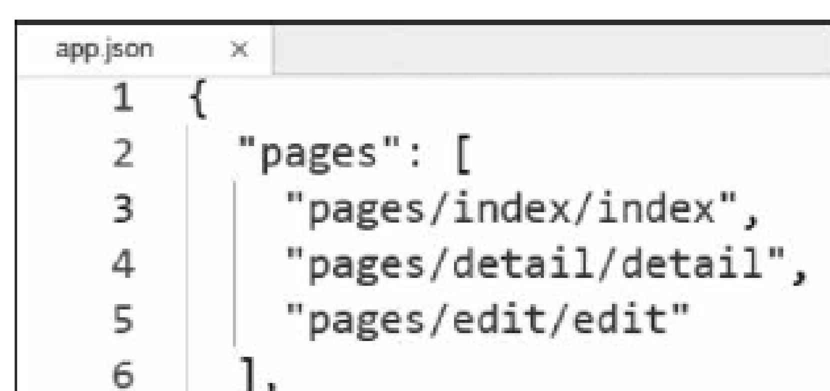


图 20-5 在 app.json 中添加页面配置的代码



图 20-6 页面文件创建完成

20.3.4　创建其他文件


视频讲解

接下来创建其他自定义文件，本项目还需要以下 3 个文件夹。

- images：用于存放图片素材，已存在；
- utils：用于存放公共 JS 文件，需要手动创建；
- vant-weapp：用于存放有赞 UI 库的 dist 目录，需要手动创建。

文件夹名称可以由开发者自定义，这里仅作为示例参考。

1 添加图片文件

本项目将用到一套动物头像卡通图标为新增好友随机生成头像，图片素材来源于网络，效果如图 20-7 所示。

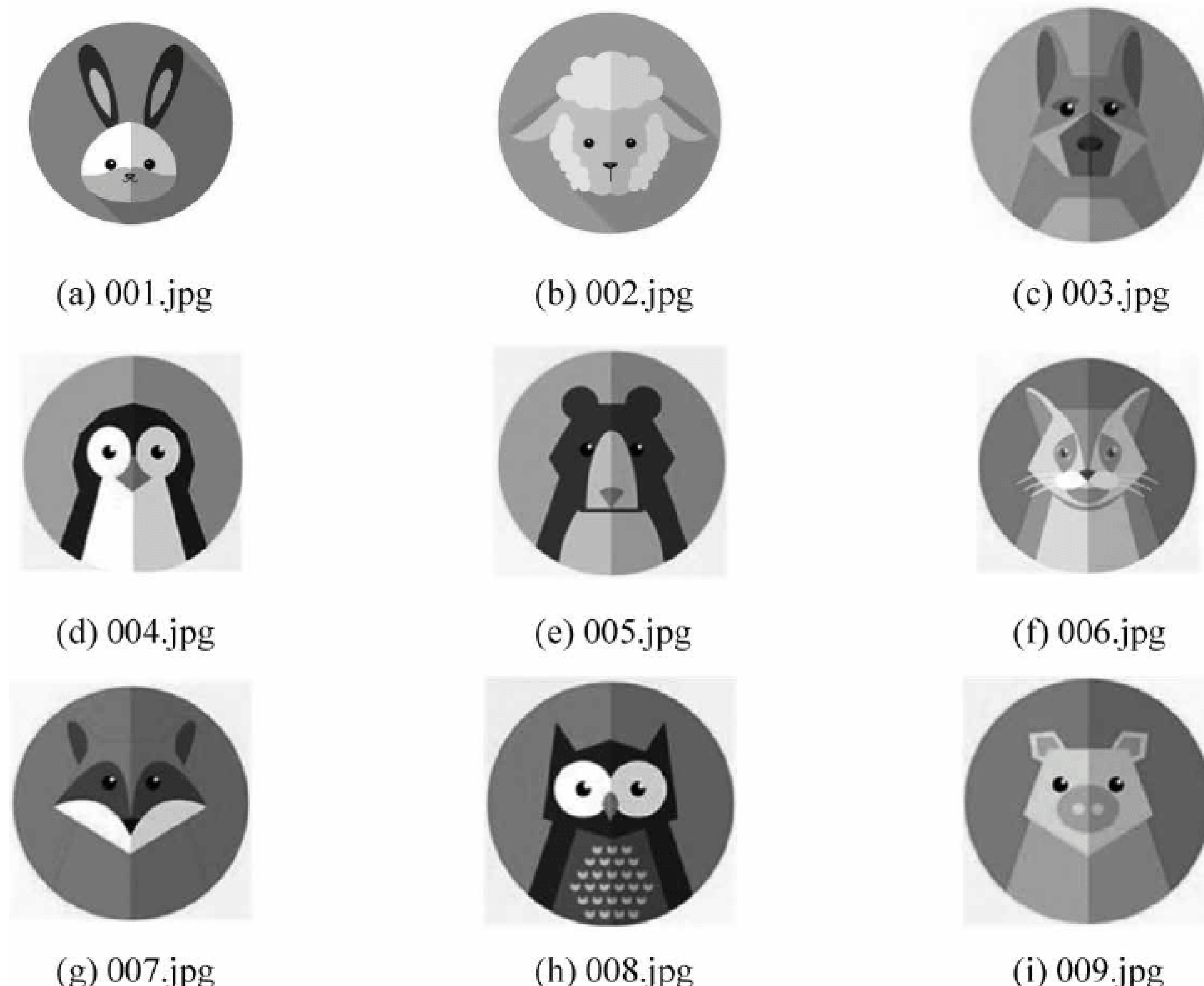


图 20-7　图标素材展示

右击目录结构中的 images 文件夹，选择"硬盘打开"，创建二级目录 avatar，并将图片复制、粘贴进去。

2 创建公共 JS 文件

右击 utils 文件夹，选择"新建"→JS，输入 utils 后按回车即创建公共文件 utils.js。

3 添加 UI 组件库

右击 vant-weapp 文件夹，选择"硬盘打开"，将事先下载好的 Vant Weapp 组件库源代码中的 dist 目录复制、粘贴进去。全部完成后的目录结构如图 20-8 所示。

此时文件配置就全部完成，20.4 节将正式进行页面布局和样式设计。

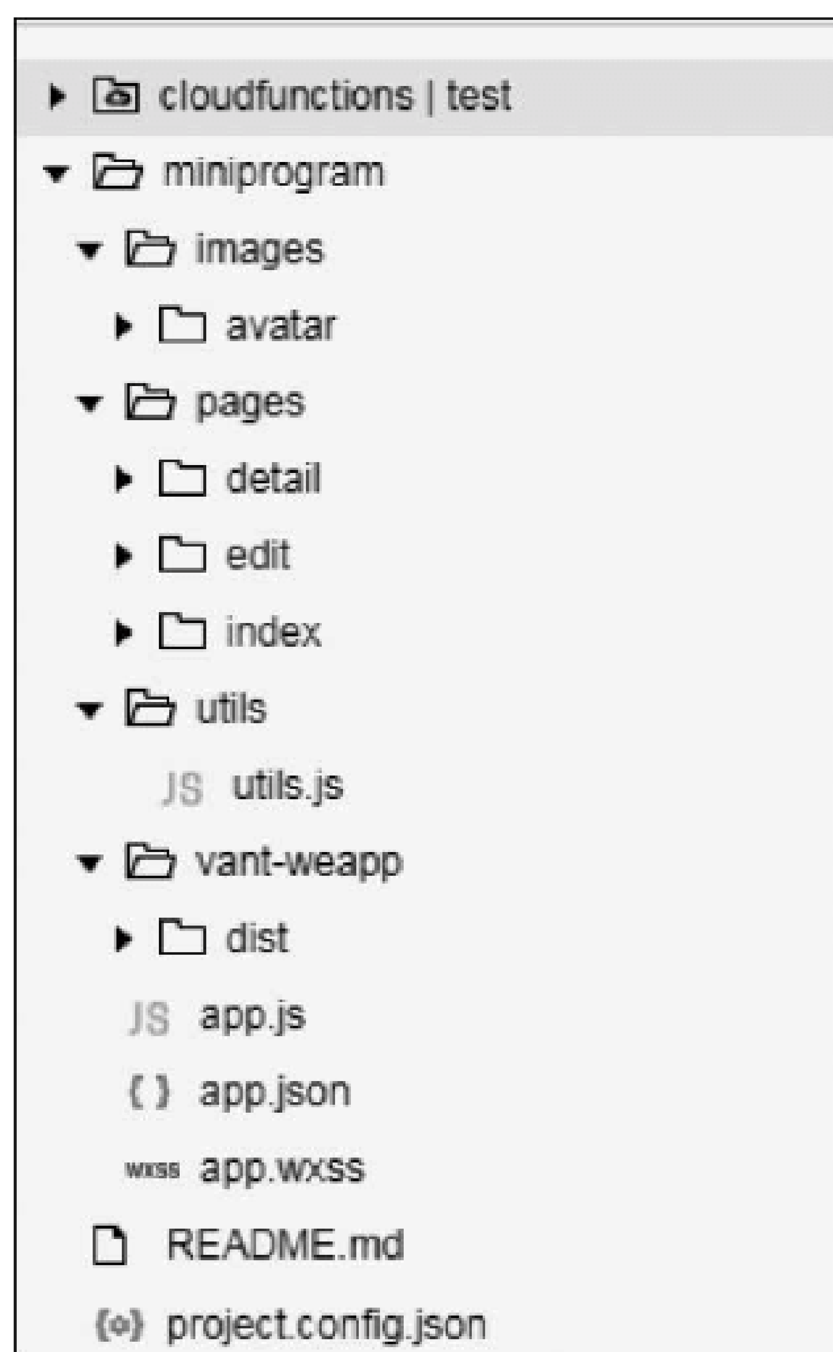


图 20-8　完整目录结构创建完成



20.4 视图设计



20.4.1 导航栏设计



视频讲解

用户可以通过在 app.json 中对 window 属性进行重新配置来自定义导航栏效果。更改后的 app.json 文件代码如下：

```
1.  {
2.    "pages": [代码略],
3.    "window": {
4.      "navigationBarBackgroundColor": "#DE6E6D",
5.      "navigationBarTitleText": "生日管家"
6.    }
7.  }
```

上述代码可以更改导航栏背景色为深粉色、字体为白色，效果如图 20-9 所示。

20.4.2 页面设计



视频讲解

1 首页设计

首页主要包含 3 部分内容，即"添加新朋友"按钮、搜索框和好友列表，页面设计如图 20-10 所示。



图 20-9 自定义导航栏效果



图 20-10 首页设计图

计划使用如下组件。

- 按钮：Vant Weapp 中的按钮组件< van-button >；
- 搜索框：Vant Weapp 中的搜索组件< van-search >；
- 好友列表：Vant Weapp 中的卡片组件< van-card >。

首先在 JSON 文件(pages/index/index.json)中添加对于第三方组件的引用声明：

```
1.  {
2.    "usingComponents": {
3.      "van-search": "/vant-weapp/dist/search/index",
4.      "van-card": "/vant-weapp/dist/card/index",
5.      "van-button": "/vant-weapp/dist/button/index"
6.    }
7.  }
```

注意： 当前示例代码表示该组件是在根目录的 vant-weapp 文件目录下的 dist 二级目录中。开发者如果变更了 UI 组件库的目录名称或路径位置，请根据实际情况填写。

WXML(pages/index/index.wxml)代码如下：

```
1.  <!-- "添加新朋友"按钮 -->
2.  <van-button block type='default'>添加新朋友</van-button>
3.
4.  <!-- 搜索框 -->
5.  <van-search placeholder="请输入搜索关键词" show-action bind:search="onSearch" bind:
cancel="onCancel" />
6.
7.  <!-- 好友列表 -->
8.  <van-card centered desc="06-23" title="小兔" thumb="/images/avatar/001.jpg">
9.      <view slot="footer">
10.     距离下个生日
11.     <text style='color:red;font-weight:bold;'>100 天</text>
12.   </view>
13. </van-card>
```

此时效果如图 20-11 所示。

这里的好友姓名、头像、生日和距离下次生日多少天都是为了演示效果临时填写的，后续会换成实际录入的好友信息，并形成列表效果。



视频讲解

2 好友信息编辑页设计

好友信息编辑页主要包含一系列表单组件和两个按钮。表单组件主要用于填写好友的姓名、性别、生日、电话和关系，两个按钮分别用于保存或取消当前登记的信息。

页面设计如图 20-12 所示。

由于暂时没有做点击跳转的逻辑设计，所以可以在开发工具顶端选择“普通编译”下的“添加编译模式”，添加界面如图 20-13 所示。

此时预览就可以直接显示 edit 页面了，其他页面也可以这么做，设计完毕后改回“普通编译”模式即可重新显示首页。

计划使用<form>组件进行整体布局，内部包含的组件如下。

- 行：Vant Weapp 中的布局组件<van-row>；
- 列：Vant Weapp 中的布局组件<van-col>；
- 文本标签：<label>组件；
- 文本输入框：<input>组件；
- 单选框：<radio>和<radio-group>组件；
- 日期选择器：<picker>组件；

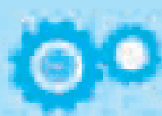




图 20-11 首页效果图



图 20-12 好友信息编辑页设计图



图 20-13 添加 edit 页面的编译模式

- 按钮：< button >组件。

首先在 JSON 文件(pages/edit/edit.json)中添加对于第三方组件的引用声明：

```
1.  {
2.    "usingComponents": {
3.      "van-row": "/vant-weapp/dist/row/index",
4.      "van-col": "/vant-weapp/dist/col/index"
5.    }
6.  }
```

在 WXML 文件(pages/edit/edit.wxml)中添加< form >组件，并在该组件内部使用 Vant Weapp 的布局组件< van-row >将页面分割成 6 行，代码如下：

```
1.  <form>
2.    <!-- 第 1 行 -->
```

```
3.     < van - row >
4.       < van - col span = "6"></van - col >
5.       < van - col span = "18"></van - col >
6.     </van - row >
7.
8.     <!-- 第 2 行 -->
9.     < van - row >
10.      < van - col span = "6"></van - col >
11.      < van - col span = "18"></van - col >
12.    </van - row >
13.
14.    <!-- 第 3 行 -->
15.    < van - row >
16.      < van - col span = "6"></van - col >
17.      < van - col span = "18"></van - col >
18.    </van - row >
19.
20.    <!-- 第 4 行 -->
21.    < van - row >
22.      < van - col span = "6"></van - col >
23.      < van - col span = "18"></van - col >
24.    </van - row >
25.
26.    <!-- 第 5 行 -->
27.    < van - row >
28.      < van - col span = "6"></van - col >
29.      < van - col span = "18"></van - col >
30.    </van - row >
31.
32.    <!-- 第 6 行 -->
33.    < van - row >
34.      < van - col span = "18" offset = "3">
35.      </van - col >
36.      < van - col span = "18" offset = "3">
37.      </van - col >
38.    </van - row >
39.
40.  </form >
```

其中前 5 行用于放置表单组件，因此在< van-row >组件内部再使用< van-col >组件进行分割。开发者需要了解的是< van-row >组件默认被平均分为 24 个栅格，因此可以尝试使用 span 属性将< van-col >再分成 6：18 的比例，其中前 6 格用于放置文本标签，后面 18 格用于放置文本输入框、单选框、日期选择器。

第 6 行用于放置两个按钮组件，对应的< van-row >组件内部使用< van-col >组件分割成 18 个栅格，并且使用 offset 属性让左边空 3 个栅格，以便按钮可以居中显示。由于第 2 个按钮所占的比例较多，会被挤到下一行单独显示。

对于上述比例，开发者可以根据自己的喜好自由修改。

然后继续在< van-col >组件中添加对应的内容，修改后的 WXML(pages/edit/edit. wxml)代码如下：

```
1.  < form >
2.    <!-- 第 1 行 -->
```

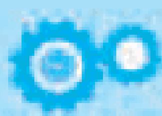
 

```
3.      < van - row >
4.        < van - col span = "6">
5.          < label >姓名</label >
6.        </van - col >
7.        < van - col span = "18">
8.          < input name = 'name' placeholder = '请输入姓名'></input >
9.        </van - col >
10.     </van - row >
11.
12.     <!-- 第 2 行 -->
13.     < van - row >
14.       < van - col span = "6">
15.         < label >性别</label >
16.       </van - col >
17.       < van - col span = "18">
18.         < radio - group name = 'gender'>
19.           < radio color = '#DE6E6D' value = '1' />男
20.           < radio color = '#DE6E6D' value = '2' />女
21.         </radio - group >
22.       </van - col >
23.     </van - row >
24.
25.     <!-- 第 3 行 -->
26.     < van - row >
27.       < van - col span = "6">
28.         < label >生日</label >
29.       </van - col >
30.       < van - col span = "18">
31.         < picker name = 'birthday' mode = 'date'>
32.           < view >{{date}}</view >
33.         </picker >
34.       </van - col >
35.     </van - row >
36.
37.     <!-- 第 4 行 -->
38.     < van - row >
39.       < van - col span = "6">
40.         < label >电话</label >
41.       </van - col >
42.       < van - col span = "18">
43.         < input name = 'tel' type = 'number' placeholder = '请输入联系电话'></input >
44.       </van - col >
45.     </van - row >
46.
47.     <!-- 第 5 行 -->
48.     < van - row >
49.       < van - col span = "6">
50.         < label >关系</label >
51.       </van - col >
52.       < van - col span = "18">
53.         < input name = 'relationship' placeholder = '描述你们的关系'></input >
54.       </van - col >
55.     </van - row >
56.
57.     <!-- 第 6 行 -->
```

```
58.     <van-row>
59.       <van-col span="18" offset="3">
60.         <button>保存记录</button>
61.       </van-col>
62.       <van-col span="18" offset="3">
63.         <button>取消修改</button>
64.       </van-col>
65.     </van-row>
66.
67. </form>
```

WXSS(pages/edit/edit.wxss)代码如下:

```
1.  /* 行布局 */
2.  van-row {
3.    margin: 20rpx 20rpx;
4.    text-align: center;
5.  }
6.
7.  /* 文本标签 */
8.  label {
9.    padding: 10rpx;
10.   color: #DE6E6D;
11.   line-height: 80rpx;
12. }
13.
14. /* 文本输入框 */
15. input {
16.   border: 1rpx solid #DE6E6D;
17.   width: 480rpx;
18.   height: 80rpx;
19.   border-radius: 20rpx;
20. }
21.
22. /* 单选框组 */
23. radio-group {
24.   width: 480rpx;
25.   line-height: 80rpx;
26. }
27.
28. /* 单选框 */
29. radio{
30.   margin: 0 20rpx;
31. }
32.
33. /* 日期选择器中的 view */
34. picker view {
35.   width: 480rpx;
36.   line-height: 80rpx;
37. }
38.
39. /* 按钮 */
40. button {
41.   margin: 20rpx;
```



```
42.     background - color: #DE6E6D;
43.     color: white;
44. }
```

为了进行样式效果的预览，还需要在 JS 文件的 data 中录入日期选择器的提示语句。

JS(pages/edit/edit.js)代码片段如下：

```
1.   Page({
2.     /**
3.       * 页面的初始数据
4.       */
5.     data: {
6.       date: '点击设置生日'
7.     },
8.   })
```

当前效果如图 20-14 所示。

由图可见，此时可以显示完整的样式效果。

3 好友信息展示页设计



视频讲解

好友信息展示页用于给用户浏览好友详细信息，需要用户点击首页的好友列表，然后在新窗口中打开该页面。好友信息展示页包括好友的头像、姓名、性别、生日、电话、关系、距离出生的天数、距离下个生日的天数，以及“修改”按钮和“删除”按钮。

页面设计如图 20-15 所示。



图 20-14　好友信息编辑页效果图

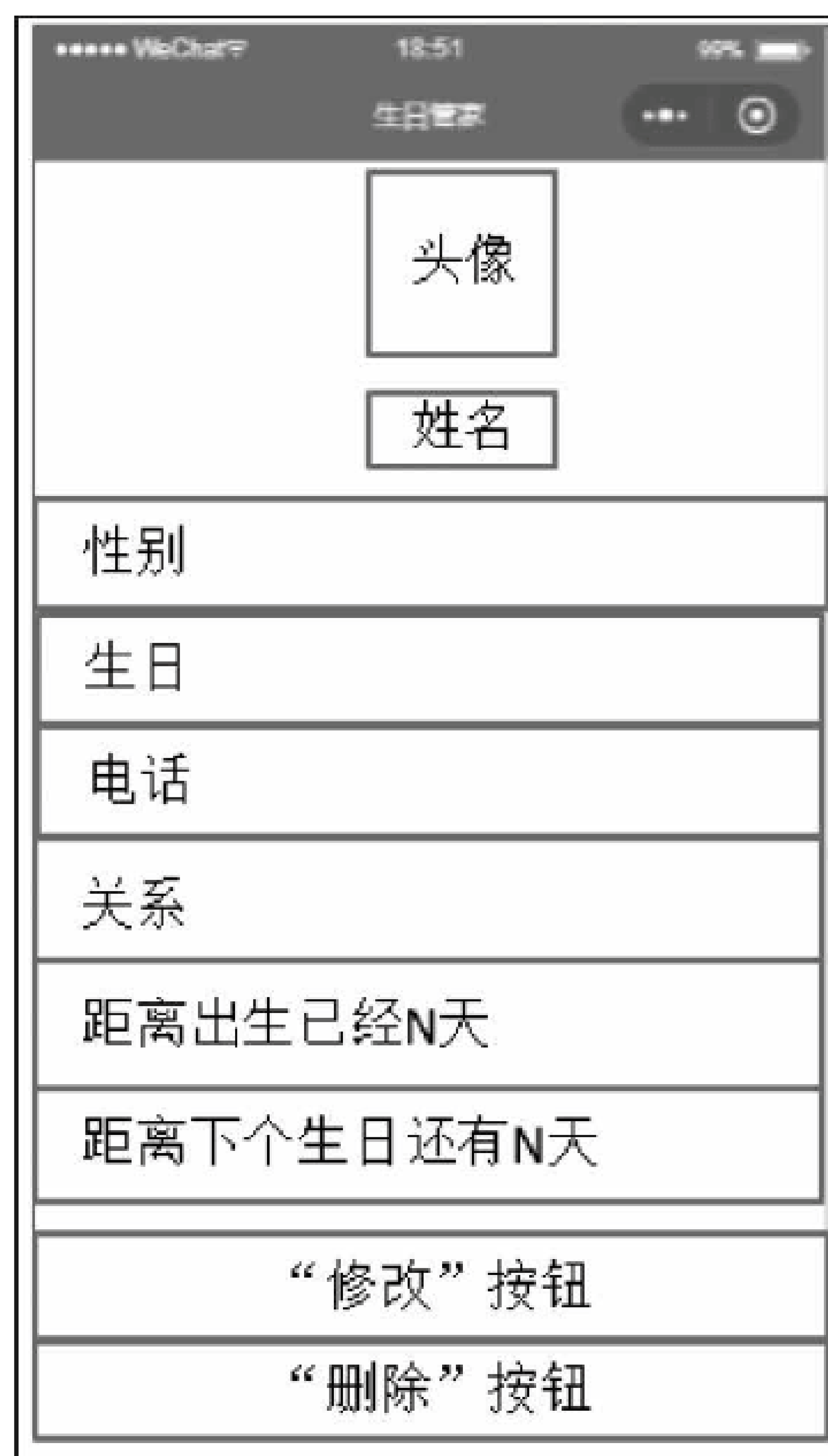


图 20-15　好友信息展示页设计图

计划将整个页面分为 3 个部分，解释如下。

- 顶端头像和姓名：<image>和<view>组件；
- 中间个人信息数据：Vant Weapp 中的单元格组件<van-cell-group>和<van-cell>；

• 底部的两个按钮：Vant Weapp 中的按钮组件< van-button >。

首先在 JSON 文件(pages/detail/detail.json)中添加对于第三方组件的引用声明：

```
1.  {
2.    "usingComponents": {
3.      "van-button": "/vant-weapp/dist/button/index",
4.      "van-cell": "/vant-weapp/dist/cell/index",
5.      "van-cell-group": "/vant-weapp/dist/cell-group/index"
6.    }
7.  }
```

WXML(pages/detail/detail.wxml)代码如下：

```
1.  <!-- 顶端头像和姓名 -->
2.  <view class='avatarBox'>
3.    <image src='/images/avatar/001.jpg'></image>
4.    <view>测试好友</view>
5.  </view>
6.
7.  <!-- 个人信息展示 -->
8.  <van-cell-group>
9.    <van-cell title="性别" value="女" />
10.   <van-cell title="生日" value="1991-10-10" />
11.   <van-cell title="电话" value="1234567" />
12.   <van-cell title="关系" value="同学" />
13.   <van-cell title="距离出生已经" value="10076 天" />
14.   <van-cell title="距离下个生日还有" value="50 天" />
15. </van-cell-group>
16.
17. <!-- 按钮区域 -->
18. <van-button block type='warning'>修改</van-button>
19. <van-button block type='danger'>删除</van-button>
```

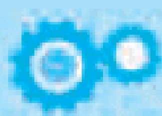
WXSS(pages/detail/detail.wxss)代码如下：

```
1.  /* 头像和姓名区域 */
2.  .avatarBox{
3.    display: flex;
4.    flex-direction: column;
5.    align-items: center;
6.  }
7.
8.  /* 头像图片 */
9.  .avatarBox image{
10.   width: 200rpx;
11.   height: 200rpx;
12.   margin: 20rpx;
13. }
14.
15. /* 姓名 */
16. .avatarBox view{
17.   margin-bottom: 50rpx;
18. }
```



图 20-16　好友信息展示页效果图

当前效果如图 20-16 所示。

由图可见，此时可以显示好友信息展示页的完整样式效果。由于尚未获得实际录入云数据库的好友信息，当前仅供作为样式参考。

此时页面布局与样式设计就已完成，20.5 节将介绍如何进行逻辑处理。



20.5 逻辑实现



20.5.1 公共逻辑



视频讲解

1 utils.js 文件逻辑

在公共 JS 文件（utils/utils.js）中添加自定义函数 getToday，用于获取当前格式化日期，相关函数代码如下：

```
1.  //获取当前格式化日期
2.  function getToday(){
3.    //获取当前日期对象
4.    var now = new Date()
5.    //获取当前年份(4 位数)
6.    var y = now.getFullYear()
7.    //获取当前月份
8.    var m = now.getMonth() + 1
9.    //获取当前日期
10.   var d = now.getDate()
11.   //格式化当天日期
12.   var today = y + '/' + m + '/' + d
13.
14.   return today
15. }
```

在公共 JS 文件（utils/utils.js）中添加自定义函数 getFullYear，用于获取当前年份（4 位数），相关函数代码如下：

```
1.  //获取当前年份(4 位数)
2.  function getFullYear() {
3.    //获取当前日期对象
4.    var now = new Date()
5.    //获取当前年份(4 位数)
6.    var y = now.getFullYear()
7.
8.    return y
9.  }
```

在公共 JS 文件（utils/utils.js）中添加自定义函数 dateDiff，用于计算两个日期之间的天数差，相关函数代码如下：

```
1.  //计算天数差
2.  function dateDiff(sDate1, sDate2) {
3.    sDate1 = sDate1.replace(/-/g, '/')
4.    sDate2 = sDate2.replace(/-/g, '/')
5.    var oDate1 = new Date(sDate1)
6.    var oDate2 = new Date(sDate2)
7.    var iDays = parseInt((oDate2 - oDate1) / 1000 / 3600 / 24)
```

```
8.     //把相差的毫秒数转换为天数
9.     return iDays
10. }
```

在公共 JS 文件(utils/utils.js)中添加自定义函数 getNextBirthday,用于计算当前日期距离下个生日还有多少天,相关函数代码如下:

```
1.  //计算距离下个生日还有多少天
2.  function getNextBirthday(b_day) {
3.     //获取当前日期
4.     var today = getToday()
5.     //获取当前年份
6.     var y = getFullYear()
7.
8.     //计算日期差
9.     var n = dateDiff(today, y + '-' + b_day)
10.
11.    //今年生日已经过完了
12.    if (n < 0) {
13.      //获得明年年份
14.      y++
15.      //计算日期差
16.      n = dateDiff(today, y + '-' + b_day)
17.    }
18.
19.    return n
20. }
```

最后需要在 utils.js 中使用 module.exports 语句暴露函数出口,代码片段如下:

```
1.  module.exports = {
2.     getToday:getToday,
3.     getFullYear:getFullYear,
4.     dateDiff: dateDiff,
5.     getNextBirthday: getNextBirthday
6.  }
```

现在就完成了公共逻辑处理的部分,还需要分别在首页(index)和好友信息展示页(detail)的 JS 文件顶端引用这个公共 JS 文件,引用代码如下:

```
var utils = require('../../utils/utils.js')            //引用公共 JS 文件
```

需要注意小程序在这里暂时还不支持绝对路径引用,只能使用相对路径。



视频讲解

2 云数据库逻辑

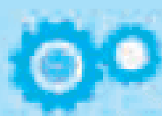
在首页(index)、好友信息编辑页(edit)和好友信息展示页(detail)的 JS 文件顶端都需要追加对云数据库的声明,相关函数代码如下:

```
1.  const db = wx.cloud.database()
2.  const birthday = db.collection('birthday')
```

此时这 3 个页面的 JS 文件均可以访问云数据库中的 birthday 数据集了。

20.5.2 好友信息编辑页逻辑

好友信息编辑页(edit)有两种进入渠道,一是点击首页的“添加新朋友”按钮打开,此时需



要录入全新的数据；二是点击好友信息展示页(detail)的“修改”按钮打开，此时需要展示当前好友的信息，并允许用户进行修改。


视频讲解

1 更新生日数据

当前的日期选择器是暂时无法将选择的日期显示在页面上的，因此需要对其进行修改。

WXML(pages/edit/edit.wxml)代码片段修改如下：

```
1.  <form>
2.    …
3.
4.    <!-- 第 3 行 -->
5.    <van-row>
6.      <van-col span="6">
7.        <label>生日</label>
8.      </van-col>
9.      <van-col span="18">
10.       <picker name='birthday' mode='date' bindchange='dateChange' value='{{date}}'>
11.         <view>{{date}}</view>
12.       </picker>
13.     </van-col>
14.   </van-row>
15.
16.   …
17. </form>
```

在 edit.js 文件的 Page()内部追加 dateChange 函数，代码片段如下：

```
1.    /**
2.     * 自定义函数 -- 更新页面上显示的出生日期
3.     */
4.    dateChange: function(e) {
5.      this.setData({
6.        date: e.detail.value
7.      })
8.    },
```

此时页面效果如图 20-17 所示。

由图可见，此时已经可以获取日期选择器中的数据并显示在页面上。接下来将分别考虑录入新朋友信息和修改已存在朋友信息两种情况。


视频讲解

2 录入新朋友信息

假设当前是点击首页的“添加新朋友”按钮跳转而来，修改 edit 页面的编译模式并携带参数 id=new 表示新朋友，如图 20-18 所示。

可以在 edit.js 文件的 onLoad 函数中获取该参数，代码片段如下：

```
1.    /**
2.     * 生命周期函数 -- 监听页面加载
3.     */
4.    onLoad: function(options) {
5.      //获取携带的参数 id
6.      let id = options.id
7.      //更新 id 数据
8.      this.setData({ id: id })
9.    },
```



(a) 点击修改生日



(b) 生日更新后的页面显示

图 20-17　生日数据更新效果图

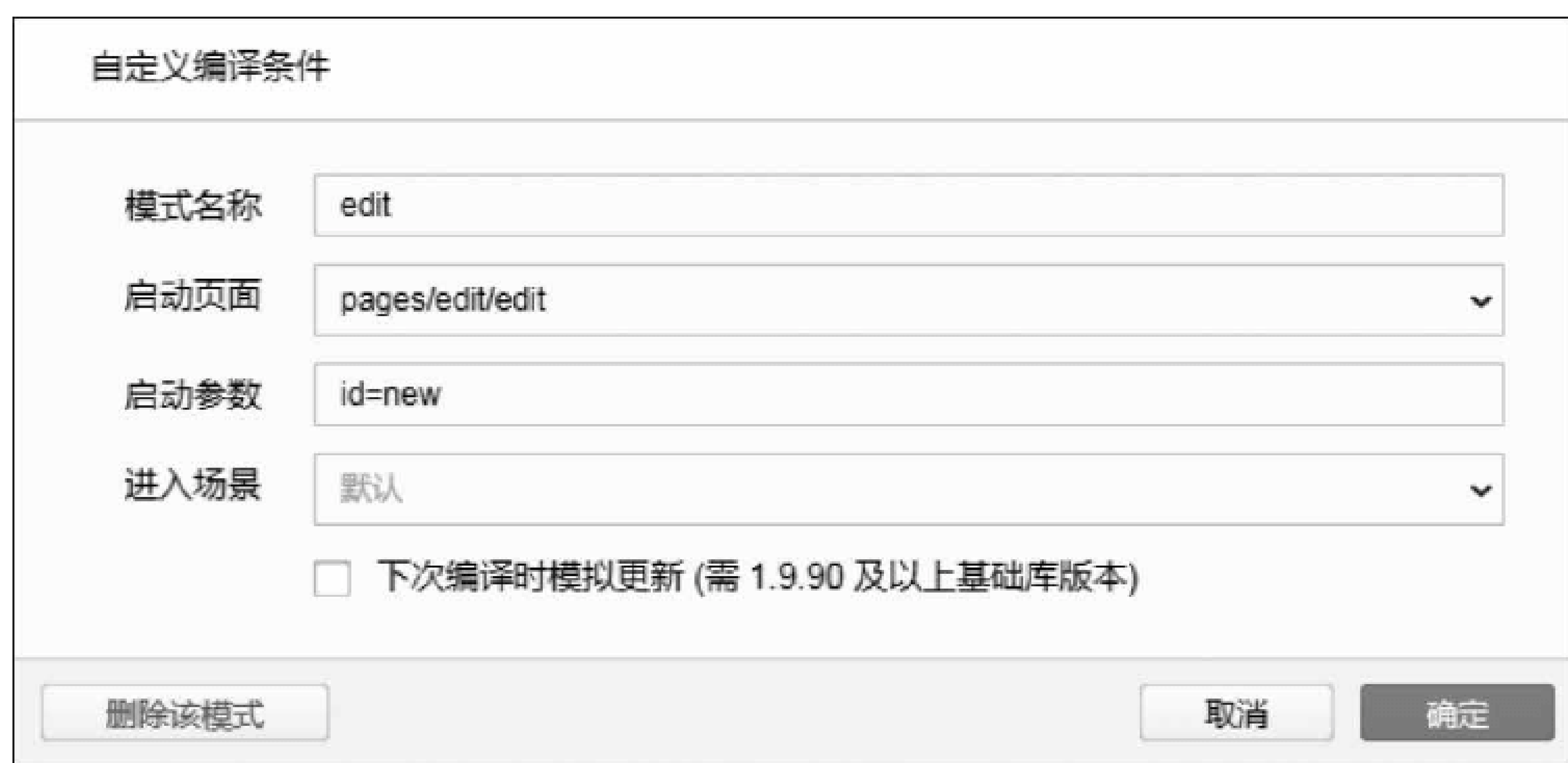


图 20-18　修改 edit 页面的编译模式

为< form >组件添加表单提交事件，并为其中的“保存记录”按钮添加 form-type 属性。WXML(pages/my/my.wxml)代码片段修改如下：

```
1.  < form bindsubmit = 'onSubmit'>
2.    …
3.
4.    <!-- 第 6 行 -->
5.    < van - row >
6.      < van - col span = "18" offset = "3">
7.        < button form - type = 'submit'>保存记录</button >
8.      </van - col >
9.      < van - col span = "18" offset = "3">
10.       < button >取消修改</button >
```

```
11.     </van-col>
12.   </van-row>
13. </form>
```

在 edit.js 文件的 Page()内部追加 onSubmit 函数，代码片段如下：

```
1.    /**
2.     * 自定义函数 -- 提交表单数据
3.     */
4.    onSubmit: function(e) {
5.      //获取表单中提交的全部数据
6.      let info = e.detail.value
7.
8.      //追加一个不带年份的生日信息
9.      let date = info.birthday.substring(5)
10.     info.date = date
11.
12.     //获取好友 id
13.     let id = this.data.id
14.
15.     //添加新朋友
16.     if(id == 'new') {
17.       //随机选择一个头像
18.       let i = Math.ceil(Math.random() * 9)
19.       info.avatar = '/images/avatar/00' + i + '.jpg'
20.
21.       //往云数据库中添加当前好友信息
22.       birthday.add({
23.         data: info,
24.         success: res => {
25.           //成功后返回首页
26.           wx.navigateBack()
27.         },
28.         fail: err => {
29.           //失败提示
30.           wx.showToast({
31.             title: '保存失败',
32.           })
33.         }
34.       })
35.     }
36.     //好友已存在
37.     else {
38.       //暂不填写
39.     }
40.   },
```

此时页面效果如图 20-19 所示。

由图可见，目前已经成功录入数据到云数据库中。开发者可以多录入几条测试数据，用于后续首页的好友列表展示。



视频讲解

3 修改当前好友信息

假设当前是点击好友展示页的“修改”按钮跳转而来，修改 edit 页面的编译模式并携带参数 id= _id 表示已录入过的朋友，如图 20-20 所示。

需要注意的是，这里的_id 属性值仅为参考效果，开发者在云数据集中可



(a) 录入测试数据　　(b) 录入成功后云数据库中的显示效果

图 20-19　录入新朋友信息

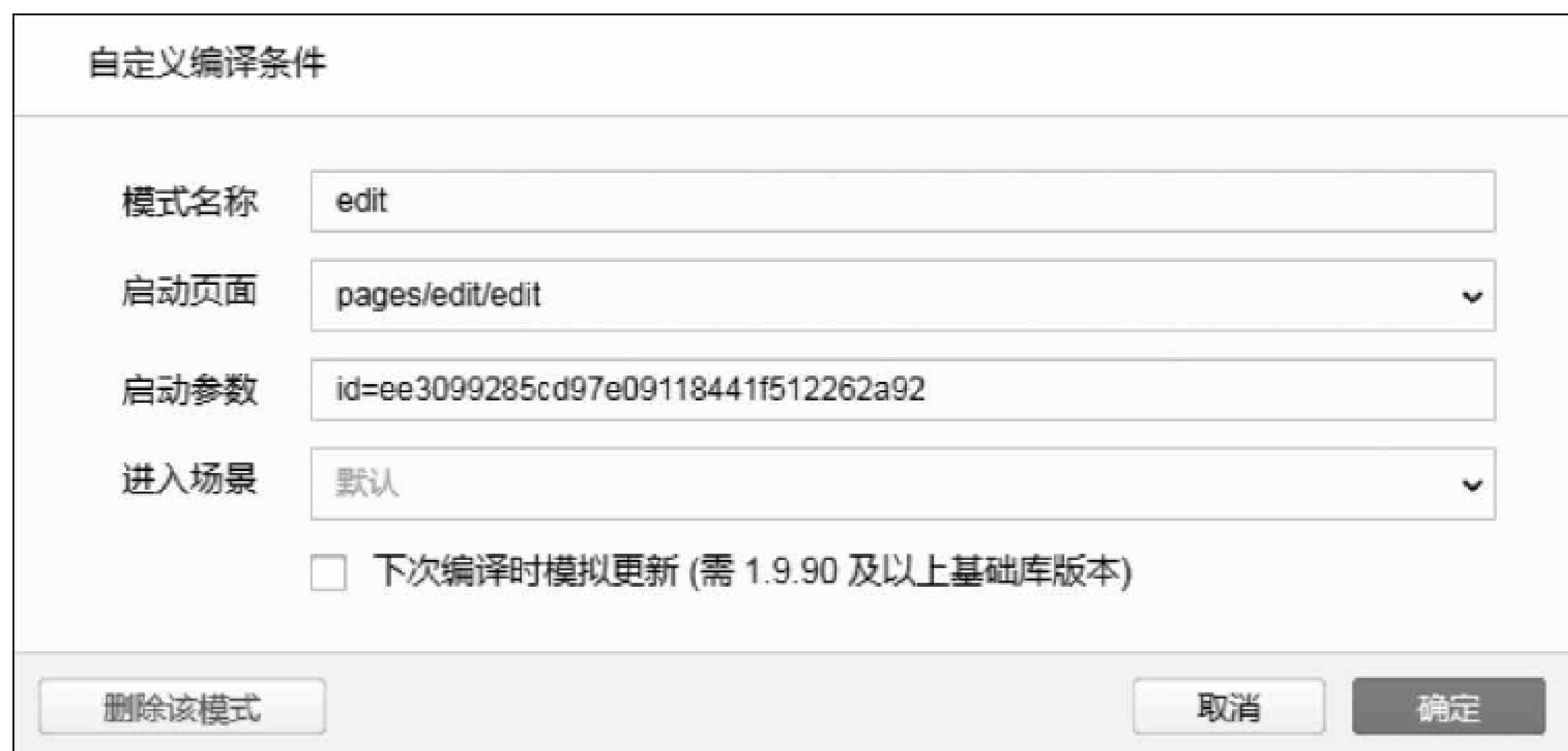


图 20-20　修改 edit 页面的编译模式

以查到，从中选择任意一条记录进行使用即可模拟跳转效果。

在 edit.js 文件的 onLoad 函数中根据 id 参数的取值获取好友信息，代码片段如下：

```
1.    /**
2.     * 生命周期函数--监听页面加载
3.     */
4.    onLoad: function(options) {
5.      //获取携带的参数 id
6.      let id = options.id
7.      //更新 id 数据
8.      this.setData({ id: id })
9.
10.     //如果好友已存在
11.     if (id != 'new') {
12.       //根据好友 id 从云数据库中获取好友信息
13.       birthday.doc(id).get({
14.         success: res => {
15.           this.setData({
16.             info: res.data,
17.             date: res.data.birthday
```

```
18.             })
19.           }
20.         })
21.     }
22.   },
```

然后将获取到的数据展示在页面中,WXML(pages/edit/edit.wxml)代码片段修改如下:

```
1.  < form bindsubmit = 'onSubmit'>
2.    <!-- 第 1 行 -->
3.    < van - row >
4.      < van - col span = "6">
5.        < label >姓名</label >
6.      </van - col >
7.      < van - col span = "18">
8.        < input name = 'name' placeholder = '请输入姓名' value = '{{info.name}}'></input >
9.      </van - col >
10.   </van - row >
11.
12.   <!-- 第 2 行 -->
13.   < van - row >
14.     < van - col span = "6">
15.       < label >性别</label >
16.     </van - col >
17.     < van - col span = "18">
18.       < radio - group name = 'gender'>
19.         < radio color = '#DE6E6D' value = '1' checked = '{{info.gender == 1}}' />男
20.         < radio color = '#DE6E6D' value = '2' checked = '{{info.gender == 2}}' />女
21.       </radio - group >
22.     </van - col >
23.   </van - row >
24.
25.   <!-- 第 3 行(代码略) -->
26.
27.   <!-- 第 4 行 -->
28.   < van - row >
29.     < van - col span = "6">
30.       < label >电话</label >
31.     </van - col >
32.     < van - col span = "18">
33.       < input name = 'tel' type = 'number' placeholder = '请输入联系电话' value = '{{info.tel}}'>
</input >
34.     </van - col >
35.   </van - row >
36.
37.   <!-- 第 5 行 -->
38.   < van - row >
39.     < van - col span = "6">
40.       < label >关系</label >
41.     </van - col >
42.     < van - col span = "18">
43.        < input name = 'relationship' placeholder = '描述你们的关系' value = '{{info.
relationship}}'></input >
44.     </van - col >
45.   </van - row >
46.
```

```
47.     <!-- 第 6 行(代码略) -->
48.  </form>
```

此时的页面已经可以显示当前的好友信息了，如图 20-21 所示。

在 edit.js 文件中修改 onSubmit 函数，补充关于好友信息修改的相关代码。



图 20-21　显示当前好友信息

对应的 JS(pages/my/my.js)代码片段如下：

```
1.     /**
2.      * 自定义函数 -- 提交表单数据
3.      */
4.     onSubmit: function(e) {
5.       …
6.
7.       //添加新朋友
8.       if (id == 'new') {
9.         …
10.      }
11.      //好友已存在
12.      else {
13.        //根据好友 id 更新数据
14.        birthday.doc(id).update({
15.          data: info,
16.          success: res => {
17.            //成功后返回上一页
18.            wx.navigateBack()
19.          },
20.          fail: err => {
21.            //失败提示
22.            wx.showToast({
23.              title: '保存失败',
24.            })
25.          }
26.        })
27.      }
28.    },
```

现在尝试修改当前好友信息并提交保存，页面效果如图 20-22 所示。

此时查看云数据库对应的记录，会看到数据已经被更新。



视频讲解

4 取消修改并返回上一页

如果不希望修改数据，可以点击“取消修改”按钮停止当前操作并返回上一页。

为“取消修改”按钮添加点击事件，WXML(pages/edit/edit.wxml)代码片段修改如下：

```
1.  <form bindsubmit = 'onSubmit'>
2.    …
3.
4.    <!-- 第 6 行 -->
5.    <van - row>
6.      <van - col span = "18" offset = "3">
7.        <button form - type = 'submit'>保存记录</button>
8.      </van - col>
9.      <van - col span = "18" offset = "3">
```

(a) 初始好友信息　　　　(b) 修改好友生日后

图 20-22　修改当前好友信息

```
10.         <button bindtap = 'cancelEdit'>取消修改</button>
11.       </van-col>
12.     </van-row>
13.   </form>
```

然后在对应的 edit.js 文件中添加 cancelEdit 函数的内容，代码片段如下：

```
1.  Page({
2.    /**
3.     * 自定义函数 -- 取消修改并返回上一页
4.     */
5.    cancelEdit: function() {
6.      wx.navigateBack()
7.    },
8.  })
```

当前还不能真的返回上一页，需要等待其他页面的对应逻辑代码完成后才可以。

20.5.3　首页逻辑

首页(index)主要有 4 个功能需要实现：

- 点击顶部的“添加新朋友”按钮可以跳转 edit 页面录入好友信息；
- 展示好友列表；
- 点击好友列表跳转新页面展示好友具体信息；
- 搜索好友关键词。



视频讲解

1 跳转添加新朋友页面

修改顶部“添加新朋友”按钮的代码，为其添加点击事件 addFriend，用于跳转新页面。

相关 WXML(pages/index/index.wxml)代码片段如下：

```
1.  <!-- "添加新朋友"按钮 -->
2.  <van-button block type='default' bindtap='addFriend'>添加新朋友</van-button>
```

在JS文件中添加自定义函数addFriend，相关JS(pages/index/index.js)代码片段如下：

```
1.  Page({
2.    /**
3.     * 自定义函数--添加好友信息
4.     */
5.    addFriend: function(options) {
6.      let id = 'new'
7.      wx.navigateTo({
8.        url: '../edit/edit?id=' + id
9.      })
10.   },
11. })
```

此时页面效果如图20-23所示。



(a) 页面初始效果



(b) 点击"添加新朋友"按钮跳转新页面

图20-23 "添加新朋友"按钮点击效果

2 展示好友列表

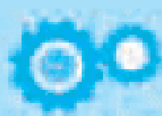


视频讲解

在JS文件中创建getFriendsList函数用于获取云数据库中的好友列表，按照出生日期升序排列，并需要对该数据进行后续处理(追加距离下个生日的天数显示)。

相关JS(pages/index/index.js)代码片段如下：

```
1.  Page({
2.    /**
3.     * 自定义函数--获取好友列表
4.     */
5.    getFriendsList: function() {
6.      //查找好友列表，按照出生日期升序排列
```



```
7.        birthday.orderBy('date', 'asc').get({
8.          success: res => {
9.            this.processData(res.data)
10.         }
11.       })
12.     },
13.
14.     /**
15.      * 自定义函数 -- 处理数据(计算距离下个生日的天数)
16.      */
17.     processData: function(list) {
18.       for (var i = 0; i < list.length; i++) {
19.         //获取不带年份的生日
20.         let date = list[i].date
21.         //计算相差几天
22.         let n = utils.getNextBirthday(date)
23.         list[i].n = n
24.       }
25.
26.       this.setData({
27.         friendsList: list
28.       })
29.     },
30.   })
```

相关 WXML(pages/index/index.wxml)代码片段修改如下：

```
1.  <!-- 好友列表 -->
2.  <block wx:for='{{friendsList}}' wx:key='{{item._id}}'>
3.    <van-card centered desc="{{(item.date)}}" title="{{item.name}}" thumb-class=
"test" thumb="{{item.avatar}}">
4.      <view slot="footer">
5.        距离下个生日
6.        <text style='color:red;font-weight:bold;'>
{{item.n}}天</text>
7.      </view>
8.    </van-card>
9.  </block>
```

此时页面效果如图 20-24 所示。



图 20-24 首页好友列表加载效果展示

最好在 JS 文件的 onShow 中也调用一下 getFriendsList 函数，以便添加新朋友后返回可以直接更新看到最新好友列表。相关 JS(pages/index/index.js)代码片段如下：

```
1.  Page({
2.    /**
3.     * 生命周期函数 -- 监听页面显示
4.     */
5.    onShow: function() {
6.      //获取好友列表
7.      this.getFriendsList()
8.    },
9.  })
```

3 点击跳转好友信息展示页


视频讲解

可以修改< van-card >组件，使得用户点击好友头像即可跳转新页面。

相关 WXML(pages/index/index.wxml)代码片段修改如下：

```
1.  <!-- 好友列表 -->
2.  <block wx:for = '{{friendsList}}' wx:key = '{{item._id}}'>
3.     <van - card centered desc = "{{(item.date)}}" title = "{{item.name}}"
thumb - link = '../detail/detail?id = {{item._id}}&n2 = {{item.n}}' thumb = "{{item.avatar}}">
4.        <view slot = "footer">
5.          距离下个生日
6.          <text style = 'color:red;font - weight:bold;'>{{item.n}}天</text>
7.        </view>
8.     </van - card>
9.  </block>
```

此时已经可以点击跳转到 detail 页面，并且成功携带了好友的 id 和距离下个生日还有 n 天这两个数据，但是仍需在 detail 页面进行携带数据的接收处理才可显示正确的内容。

4 关键词搜索功能


视频讲解

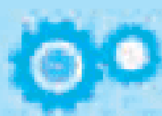
在 JS 文件中添加关键词组件< van-search >的 onSearch 和 onCancel 事件的具体内容。

相关 JS(pages/index/index.js)代码片段修改如下：

```
1.     /**
2.      * 自定义函数 -- 取消搜索
3.      */
4.     onCancel: function(e) {
5.       //获取好友列表
6.       this.getFriendsList()
7.     },
8.
9.     /**
10.     * 自定义函数 -- 搜索关键词
11.     */
12.    onSearch: function(e) {
13.      //获取搜索关键词
14.      let keyword  =  e.detail
15.
16.      //使用正则表达式模糊查询
17.      birthday.where({
18.        name: db.RegExp({
19.          regexp: keyword,
20.          options: 'i',
21.        })
22.      }).orderBy('date', 'asc').get({
23.        success: res  => {
24.          this.processData(res.data)
25.        }
26.      })
27.    },
```

此时在首页搜索框中尝试输入关键词进行搜索，会看到筛选后符合条件的好友列表。

运行效果如图 20-25 所示。





(a) 输入关键词　　(b) 搜索结果　　(c) 取消搜索

图 20-25　关键词搜索效果展示

20.5.4　好友信息展示页逻辑

好友信息展示页(detail)主要有两个功能需要实现，一是显示对应新闻；二是点击新闻底部按钮可以添加/取消新闻收藏。

1 显示当前好友信息



视频讲解

假设当前是点击首页好友列表中的某个好友头像跳转而来，修改 detail 页面的编译模式并携带参数 id 和 n2 分别表示当前好友的_id 以及距离下个生日的天数，如图 20-26 所示。

需要注意的是，这里的 id 和 n2 属性值仅为参考效果。其中 id 取值可以由开发者在云数据集中查到，从中选择任意一条记录进行使用即可模拟跳转效果。

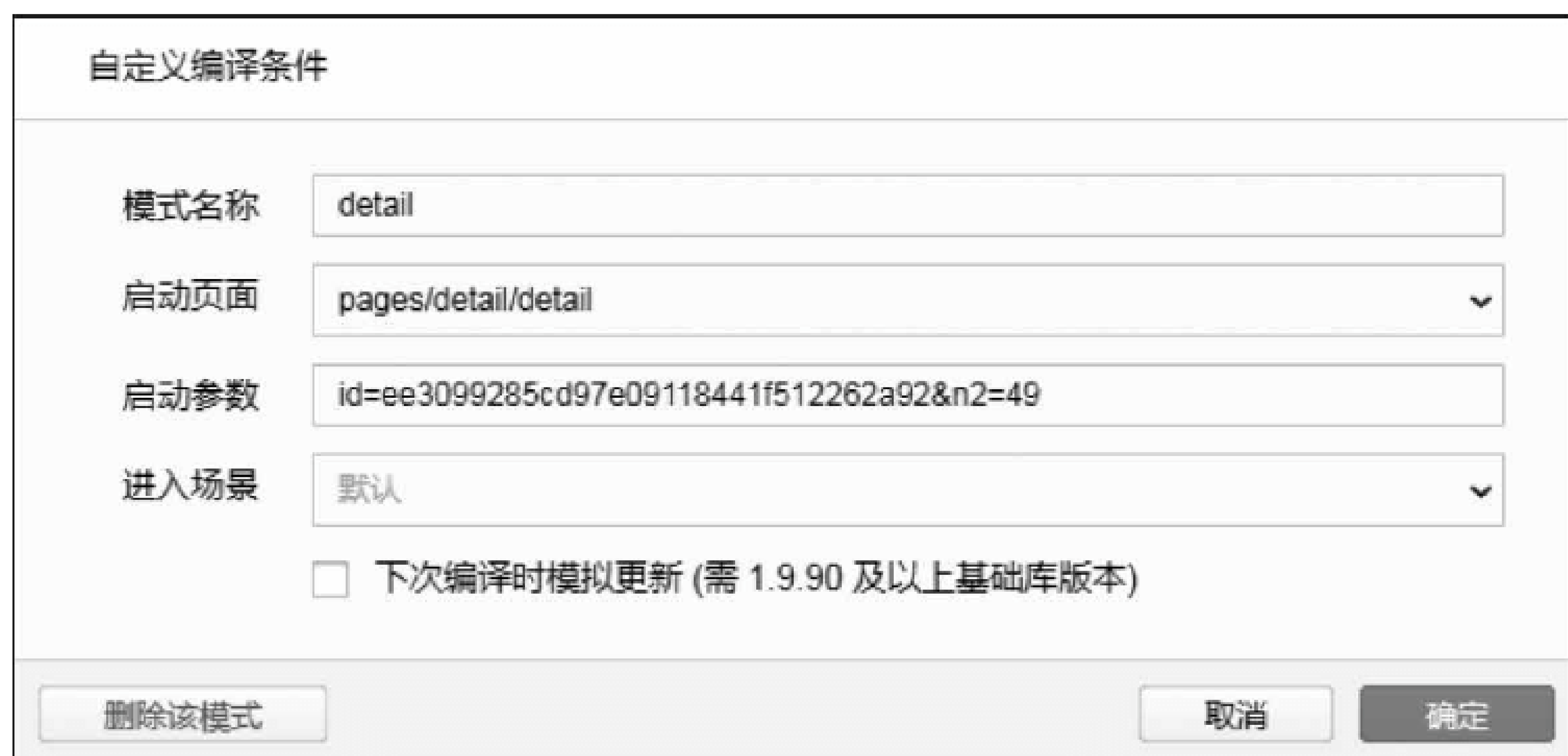


图 20-26　修改 detail 页面的编译模式

可以在 JS 文件的 onLoad 函数中分别获取从首页传来的参数 id 和 n2，并更新到页面数据中。相关 JS(pages/detail/detail.js)代码片段如下：

```
1.  Page({
2.     /**
3.      * 生命周期函数--监听页面加载
4.      */
5.    onLoad: function(options) {
6.      //获取从首页传来的参数
7.      let id = options.id                //好友 id
8.      let n2 = options.n2                //距离下个生日的天数
9.
10.     //更新页面数据
11.     this.setData({
12.       id: id,
13.       n2: n2
14.     })
15.   },
16. })
```

然后在 JS 文件的 onShow 函数中根据好友 id 查找云数据库，获取当前好友信息。相关 JS(pages/detail/detail.js)代码片段如下：

```
1.  Page({
2.    /**
3.      * 生命周期函数--监听页面显示
4.      */
5.    onShow: function() {
6.      //获取当前好友 id
7.      let id = this.data.id
8.
9.      //从云数据库中查找当前好友信息
10.     birthday.doc(id).get({
11.       success: res => {
12.         //获取当前日期
13.         let today = utils.getToday()
14.
15.         //获取生日(带年份)
16.         let b_day = res.data.birthday
17.
18.         //计算距离出生的天数
19.         let n1 = utils.dateDiff(b_day, today)
20.
21.         //更新页面数据
22.         this.setData({
23.           info: res.data,
24.           n1: n1
25.         })
26.       }
27.     })
28.   },
29. })
```

最后修改 WXML 页面，相关 WXML(pages/detail/detail.wxml)代码片段如下：

```
1.  <!-- 顶端头像和姓名 -->
2.  <view class='avatarBox'>
3.    <image src='{{info.avatar}}'></image>
4.    <view>{{info.name}}</view>
5.  </view>
```

```
6.
7.  <!-- 个人信息展示 -->
8.  <van-cell-group>
9.    <van-cell title="性别" value="{{info.gender==1?'男':'女'}}" />
10.   <van-cell title="生日" value="{{info.birthday}}" />
11.   <van-cell title="电话" value="{{info.tel}}" />
12.   <van-cell title="关系" value="{{info.relationship}}" />
13.   <van-cell title="距离出生已经" value="{{n1}}天" />
14.   <van-cell title="距离下个生日还有" value="{{n2}}天" />
15. </van-cell-group>
```

此时重新从首页点击好友头像跳转就可以发现已经能够正确显示对应的好友信息了。运行效果如图 20-27 所示。



(a) 首页好友列表



(b) 点击好友头像查看详情

图 20-27 在首页点击好友头像跳转显示好友详情的效果

2 修改好友信息



视频讲解

首先要为"修改"按钮添加点击事件，点击后跳转好友信息编辑页。

WXML(pages/detail/detail.wxml)代码片段修改如下：

```
1.  <!-- 按钮区域 -->
2.  <van-button block type='warning' bindtap='editFriend'>修改</van-button
>
3.  <van-button block type='danger'>删除</van-button>
```

在 detail.js 文件中添加 editFriend 函数，用于携带好友 ID 信息并跳转好友信息编辑页。对应 JS(pages/detail/detail.js)中的 itFriend 函数的代码片段修改如下：

```
1.  Page({
2.    /**
3.     * 自定义函数 -- 编辑好友信息
```

```
4.      * /
5.    editFriend: function() {
6.      //获取当前好友 id
7.      let id = this.data.id
8.      //跳转到编辑页面并携带参数 id
9.      wx.navigateTo({
10.       url: '../edit/edit?id = ' + id
11.     })
12.   },
13. })
```

现在尝试点击“修改”按钮，可以成功携带好友 ID 跳转到好友编辑页(edit)。此时页面效果如图 20-28 所示。



(a) 好友信息展示页



(b) 点击“修改”按钮跳转编辑页

图 20-28 好友信息展示页中“修改”按钮的使用效果

3 删除当前好友



视频讲解

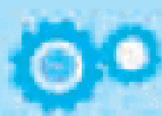
首先要为“删除”按钮添加点击事件，点击后直接从云数据库中删除此好友记录。

WXML(pages/detail/detail.wxml)代码片段修改如下：

```
1.  <!-- 按钮区域 -->
2.  < van - button block type = 'warning' bindtap = 'editFriend'>修改</van - button >
3.  < van - button block type = 'danger' bindtap = 'deleteFriend'>删除</van - button >
```

在 detail.js 文件中添加 deleteFriend 函数，用于根据好友 ID 删除指定记录。对应 JS(pages/detail/detail.js)中的 deleteFriend 函数的代码片段修改如下：

```
1.  Page({
2.    / **
3.      * 自定义函数 -- 删除好友
4.      * /
5.    deleteFriend: function() {
```



```
6.        //获取当前好友 id
7.        let id = this.data.id
8.
9.        //删除当前好友
10.       birthday.doc(id).remove({
11.         success: res => {
12.           //删除成功后返回上一页
13.           wx.navigateBack()
14.         }
15.       })
16.     },
17.   })
```

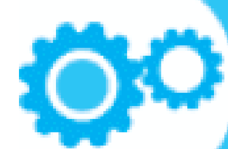
此时重新从首页点击任意好友头像跳转到好友信息展示页，然后尝试点击“删除”按钮即可删除当前好友信息并返回首页了。

运行效果如图 20-29 所示。



(a) 首页初始效果　　(b) 好友信息展示页　　(c) 删除好友后的首页效果

图 20-29　删除指定好友效果展示



20.6　完整代码展示



20.6.1　应用文件代码展示

app.json 文件的完整代码如下：

```
1.  {
2.    "pages": [
3.      "pages/index/index",
4.      "pages/detail/detail",
5.      "pages/edit/edit"
6.    ],
7.    "window": {
8.      "navigationBarBackgroundColor": "#DE6E6D",
```

```
9.        "navigationBarTitleText": "生日管家"
10.     }
11.  }
```

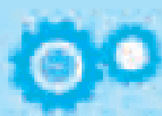
app.js 文件的完整代码如下(创建项目时自动生成):

```
1.   App({
2.     onLaunch: function() {
3.
4.       if (!wx.cloud) {
5.         console.error('请使用 2.2.3 或以上的基础库以使用云能力')
6.       } else {
7.         wx.cloud.init({
8.           traceUser: true,
9.         })
10.      }
11.
12.      this.globalData = {}
13.    }
14.  })
```

20.6.2　公共函数文件代码展示

JS 文件(utils/utils.js)的完整代码如下:

```
1.   //获取当前格式化日期
2.   function getToday(){
3.     //获取当前日期对象
4.     var now = new Date()
5.     //获取当前年份(4 位数)
6.     var y = now.getFullYear()
7.     //获取当前月份
8.     var m = now.getMonth() + 1
9.     //获取当前日期
10.    var d = now.getDate()
11.    //格式化当天日期
12.    var today = y + '/' + m + '/' + d
13.
14.    return today
15.  }
16.
17.  //获取当前年份(4 位数)
18.  function getFullYear() {
19.    //获取当前日期对象
20.    var now = new Date()
21.    //获取当前年份(4 位数)
22.    var y = now.getFullYear()
23.
24.    return y
25.  }
26.
27.  //计算天数差
28.  function dateDiff(sDate1, sDate2) {
29.    sDate1 = sDate1.replace(/-/g, '/')
30.    sDate2 = sDate2.replace(/-/g, '/')
```

```
31.   var oDate1 = new Date(sDate1)
32.   var oDate2 = new Date(sDate2)
33.   var iDays = parseInt((oDate2 - oDate1) / 1000 / 3600 / 24)
34.   //把相差的毫秒数转换为天数
35.   return iDays
36. }
37.
38. //计算距离下个生日还有多少天
39. function getNextBirthday(b_day) {
40.   //获取当前日期
41.   var today = getToday()
42.   //获取当前年份
43.   var y = getFullYear()
44.
45.   //计算日期差
46.   var n = dateDiff(today, y + '-' + b_day)
47.
48.   //今年生日已经过完了
49.   if (n < 0) {
50.     //获得明年年份
51.     y++
52.     //计算日期差
53.     n = dateDiff(today, y + '-' + b_day)
54.   }
55.
56.   return n
57. }
58.
59. module.exports = {
60.   getToday:getToday,
61.   getFullYear:getFullYear,
62.   dateDiff: dateDiff,
63.   getNextBirthday: getNextBirthday
64. }
```

20.6.3 页面文件代码展示

1 首页代码展示

JSON 文件(pages/index/index.json)的完整代码如下：

```
1.  {
2.    "usingComponents": {
3.      "van-search": "/vant-weapp/dist/search/index",
4.      "van-card": "/vant-weapp/dist/card/index",
5.      "van-button": "/vant-weapp/dist/button/index"
6.    }
7.  }
```

WXML 文件(pages/index/index.wxml)的完整代码如下：

```
1.  <!-- "添加新朋友"按钮 -->
2.  <van-button block type='default' bindtap='addFriend'>添加新朋友</van-button>
3.
4.  <!-- 搜索框 -->
```

```
5.  < van - search placeholder = "请输入搜索关键词" show - action bind:search = "onSearch" bind:
cancel = "onCancel" />
6.
7.  <!-- 好友列表 -->
8.  < block wx:for = '{{friendsList}}' wx:key = '{{item._id}}'>
9.    < van - card centered desc = "{{(item.date)}}" title = "{{item.name}}" thumb - link = '../
detail/detail?id = {{item._id}}&n2 = {{item.n}}' thumb = "{{item.avatar}}">
10.     < view slot = "footer">
11.       距离下个生日
12.       < text style = 'color:red;font - weight:bold;'>{{item.n}}天</text >
13.     </view >
14.   </van - card >
15. </block >
```

JS 文件(pages/index/index.js)的完整代码如下：

```
1.  var utils = require('../../utils/utils.js')
2.  const db = wx.cloud.database()
3.  const birthday = db.collection('birthday')
4.
5.  Page({
6.    /**
7.     * 自定义函数 -- 添加好友信息
8.     */
9.    addFriend: function(options) {
10.     let id = 'new'
11.     wx.navigateTo({
12.       url: '../edit/edit?id = ' + id
13.     })
14.   },
15.
16.   /**
17.    * 自定义函数 -- 取消搜索
18.    */
19.   onCancel: function(e) {
20.     //获取好友列表
21.     this.getFriendsList()
22.   },
23.
24.   /**
25.    * 自定义函数 -- 搜索关键词
26.    */
27.   onSearch: function(e) {
28.     //获取搜索关键词
29.     let keyword = e.detail
30.
31.     //使用正则表达式模糊查询
32.     birthday.where({
33.       name: db.RegExp({
34.         regexp: keyword,
35.         options: 'i',
36.       })
37.     }).orderBy('date', 'asc').get({
38.       success: res => {
39.         this.processData(res.data)
40.       }
```

```
41.       })
42.     },
43.
44.     /**
45.      * 自定义函数--获取好友列表
46.      */
47.     getFriendsList: function() {
48.       //查找好友列表,按照出生日期升序排列
49.       birthday.orderBy('date', 'asc').get({
50.         success: res => {
51.           this.processData(res.data)
52.         }
53.       })
54.     },
55.
56.     /**
57.      * 自定义函数--处理数据(计算距离下个生日的天数)
58.      */
59.     processData: function(list) {
60.       for (var i = 0; i < list.length; i++) {
61.         //获取不带年份的生日
62.         let date = list[i].date
63.         //计算相差几天
64.         let n = utils.getNextBirthday(date)
65.         list[i].n = n
66.       }
67.
68.       this.setData({
69.         friendsList: list
70.       })
71.     },
72.
73.     /**
74.      * 生命周期函数--监听页面显示
75.      */
76.     onShow: function() {
77.       //获取好友列表
78.       this.getFriendsList()
79.     },
80.   })
```

2 好友信息编辑页代码展示

JSON 文件(pages/edit/edit.json)的完整代码如下：

```
1.  {
2.    "usingComponents": {
3.      "van-row": "/vant-weapp/dist/row/index",
4.      "van-col": "/vant-weapp/dist/col/index"
5.    }
6.  }
```

WXML 文件(pages/edit/edit.wxml)的完整代码如下：

```
1.  <form bindsubmit='onSubmit'>
2.    <!-- 第1行 -->
3.    <van-row>
```

```
4.        < van - col span = "6">
5.          < label >姓名</label >
6.        </van - col >
7.        < van - col span = "18">
8.          < input name = 'name' placeholder = '请输入姓名' value = '{{info.name}}'></input >
9.        </van - col >
10.     </van - row >
11.
12.     <!-- 第 2 行 -->
13.     < van - row >
14.       < van - col span = "6">
15.         < label >性别</label >
16.       </van - col >
17.       < van - col span = "18">
18.         < radio - group name = 'gender'>
19.           < radio color = '#DE6E6D' value = '1' checked = '{{info.gender == 1}}' />男
20.           < radio color = '#DE6E6D' value = '2' checked = '{{info.gender == 2}}' />女
21.         </radio - group >
22.       </van - col >
23.     </van - row >
24.
25.     <!-- 第 3 行 -->
26.     < van - row >
27.       < van - col span = "6">
28.         < label >生日</label >
29.       </van - col >
30.       < van - col span = "18">
31.         < picker name = 'birthday' mode = 'date' bindchange = 'dateChange' value = '{{date}}'>
32.           < view >{{date}}</view >
33.         </picker >
34.       </van - col >
35.     </van - row >
36.
37.     <!-- 第 4 行 -->
38.     < van - row >
39.       < van - col span = "6">
40.         < label >电话</label >
41.       </van - col >
42.       < van - col span = "18">
43.         < input name = 'tel' type = 'number' placeholder = '请输入联系电话' value = '{{info.tel}}'>
</input >
44.       </van - col >
45.     </van - row >
46.
47.     <!-- 第 5 行 -->
48.     < van - row >
49.       < van - col span = "6">
50.         < label >关系</label >
51.       </van - col >
52.       < van - col span = "18">
53.         < input name = 'relationship' placeholder = '描述你们的关系' value = '{{info.
relationship}}'></input >
54.       </van - col >
55.     </van - row >
56.
```

```
57.     <!-- 第 6 行 -->
58.     <van-row>
59.       <van-col span="18" offset="3">
60.         <button form-type='submit'>保存记录</button>
61.       </van-col>
62.       <van-col span="18" offset="3">
63.         <button bindtap='cancelEdit'>取消修改</button>
64.       </van-col>
65.     </van-row>
66.
67.   </form>
```

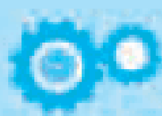
WXSS 文件(pages/edit/edit.wxss)的完整代码如下：

```
1.  /* 行布局 */
2.  van-row {
3.    margin: 20rpx 20rpx;
4.    text-align: center;
5.  }
6.
7.  /* 文本标签 */
8.  label {
9.    padding: 10rpx;
10.   color: #DE6E6D;
11.   line-height: 80rpx;
12. }
13.
14. /* 文本输入框 */
15. input {
16.   border: 1rpx solid #DE6E6D;
17.   width: 480rpx;
18.   height: 80rpx;
19.   border-radius: 20rpx;
20. }
21.
22. /* 单选框组 */
23. radio-group {
24.   width: 480rpx;
25.   line-height: 80rpx;
26. }
27.
28. /* 单选框 */
29. radio{
30.   margin: 0 20rpx;
31. }
32.
33. /* 日期选择器中的 view */
34. picker view {
35.   width: 480rpx;
36.   line-height: 80rpx;
37. }
38.
39. /* 按钮 */
40. button {
41.   margin: 20rpx;
42.   background-color: #DE6E6D;
```

```
43.   color: white;
44. }
```

JS 文件(pages/edit/edit.js)的完整代码如下：

```
1.  const db = wx.cloud.database()
2.  const birthday = db.collection('birthday')
3.
4.  Page({
5.    /**
6.     * 页面的初始数据
7.     */
8.    data: {
9.      date: '点击设置生日'
10.   },
11.
12.   /**
13.    * 自定义函数--更新页面上显示的出生日期
14.    */
15.   dateChange: function(e) {
16.     this.setData({
17.       date: e.detail.value
18.     })
19.   },
20.
21.   /**
22.    * 自定义函数--提交表单数据
23.    */
24.   onSubmit: function(e) {
25.     //获取表单中提交的全部数据
26.     let info = e.detail.value
27.
28.     //追加一个不带年份的生日信息
29.     let date = info.birthday.substring(5)
30.     info.date = date
31.
32.     //获取好友 id
33.     let id = this.data.id
34.
35.     //添加新朋友
36.     if (id == 'new') {
37.       //随机选择一个头像
38.       let i = Math.ceil(Math.random() * 9)
39.       info.avatar = '/images/avatar/00' + i + '.jpg'
40.
41.       //往云数据库中添加当前好友信息
42.       birthday.add({
43.         data: info,
44.         success: res => {
45.           //成功后返回首页
46.           wx.navigateBack()
47.         },
48.         fail: err => {
49.           //失败提示
50.           wx.showToast({
51.             title: '保存失败',
```

```
52.             })
53.           }
54.         })
55.       }
56.       //好友已存在
57.       else {
58.         //根据好友 id 更新数据
59.         birthday.doc(id).update({
60.           data: info,
61.           success: res => {
62.             //成功后返回上一页
63.             wx.navigateBack()
64.           },
65.           fail: err => {
66.             //失败提示
67.             wx.showToast({
68.               title: '保存失败',
69.             })
70.           }
71.         })
72.       }
73.     },
74. 
75.     /**
76.      * 自定义函数 -- 取消修改并返回上一页
77.      */
78.     cancelEdit: function() {
79.       wx.navigateBack()
80.     },
81. 
82.     /**
83.      * 生命周期函数 -- 监听页面加载
84.      */
85.     onLoad: function(options) {
86.       //获取携带的参数 id
87.       let id = options.id
88.       //更新 id 数据
89.       this.setData({
90.         id: id
91.       })
92. 
93.       //如果好友已存在
94.       if (id != 'new') {
95.         //根据好友 id 从云数据库中获取好友信息
96.         birthday.doc(id).get({
97.           success: res => {
98.             this.setData({
99.               info: res.data,
100.              date: res.data.birthday
101.            })
102.          }
103.        })
104.      }
105.    },
106.})
```

3 好友信息展示页代码展示

JSON 文件(pages/detail/detail.json)的完整代码如下：

```
1.  {
2.    "usingComponents": {
3.      "van-button": "/vant-weapp/dist/button/index",
4.      "van-cell": "/vant-weapp/dist/cell/index",
5.      "van-cell-group": "/vant-weapp/dist/cell-group/index"
6.    }
7.  }
```

WXML 文件(pages/detail/detail.wxml)的完整代码如下：

```
1.  <!-- 顶端头像和姓名 -->
2.  <view class='avatarBox'>
3.    <image src='{{info.avatar}}'></image>
4.    <view>{{info.name}}</view>
5.  </view>
6.
7.  <!-- 个人信息展示 -->
8.  <van-cell-group>
9.    <van-cell title="性别" value="{{info.gender==1?'男':'女'}}" />
10.   <van-cell title="生日" value="{{info.birthday}}" />
11.   <van-cell title="电话" value="{{info.tel}}" />
12.   <van-cell title="关系" value="{{info.relationship}}" />
13.   <van-cell title="距离出生已经" value="{{n1}}天" />
14.   <van-cell title="距离下个生日还有" value="{{n2}}天" />
15. </van-cell-group>
16.
17. <!-- 按钮区域 -->
18. <van-button block type='warning' bindtap='editFriend'>修改</van-button>
19. <van-button block type='danger' bindtap='deleteFriend'>删除</van-button>
```

WXSS 文件(pages/detail/detail.wxss)的完整代码如下：

```
1.  /*头像和姓名区域*/
2.  .avatarBox{
3.    display: flex;
4.    flex-direction: column;
5.    align-items: center;
6.  }
7.
8.  /*头像图片*/
9.  .avatarBox image{
10.   width: 200rpx;
11.   height: 200rpx;
12.   margin: 20rpx;
13. }
14.
15. /*姓名*/
16. .avatarBox view{
17.   margin-bottom: 50rpx;
18. }
```

JS 文件(pages/detail/detail.js)的完整代码如下：

```
1.  const utils = require('../../utils/utils.js')
2.  const db = wx.cloud.database()
3.  const birthday = db.collection('birthday')
4.
5.  Page({
6.    /**
7.     * 自定义函数--编辑好友信息
8.     */
9.    editFriend: function() {
10.     //获取当前好友 id
11.     let id = this.data.id
12.     //跳转到编辑页面并带参数 id
13.     wx.navigateTo({
14.       url: '../edit/edit?id=' + id
15.     })
16.   },
17.
18.   /**
19.    * 自定义函数--删除好友
20.    */
21.   deleteFriend: function() {
22.     //获取当前好友 id
23.     let id = this.data.id
24.
25.     //删除当前好友
26.     birthday.doc(id).remove({
27.       success: res => {
28.         //删除成功后返回上一页
29.         wx.navigateBack()
30.       }
31.     })
32.   },
33.
34.   /**
35.    * 生命周期函数--监听页面加载
36.    */
37.   onLoad: function(options) {
38.     //获取从首页传来的参数
39.     let id = options.id              //好友 id
40.     let n2 = options.n2              //距离下个生日的天数
41.
42.     //更新页面数据
43.     this.setData({
44.       id: id,
45.       n2: n2
46.     })
47.   },
48.
49.   /**
50.    * 生命周期函数--监听页面显示
51.    */
52.   onShow: function() {
53.     //获取当前好友 id
54.     let id = this.data.id
55.
```

```
56.      //从云数据库中查找当前好友信息
57.      birthday.doc(id).get({
58.        success: res => {
59.          //获取当前日期
60.          let today = utils.getToday()
61.          //获取生日(带年份)
62.          let b_day = res.data.birthday
63.          //计算距离出生的天数
64.          let n1 = utils.dateDiff(b_day, today)
65.
66.          //更新页面数据
67.          this.setData({
68.            info: res.data,
69.            n1: n1
70.          })
71.        }
72.      })
73.    },
74.  })
```

附录A APPENDIX A

服务器部署



A.1 服务器域名配置



每一个小程序在与指定域名地址进行网络通信前都必须将该域名地址添加到管理员后台白名单中。

A.1.1 配置流程

小程序开发者登录 mp. weixin. qq. com 进入管理员后台，单击“设置”下的“开发设置”，在“服务器域名”下添加或修改需要进行网络通信的服务器域名地址，如图 A-1 所示。

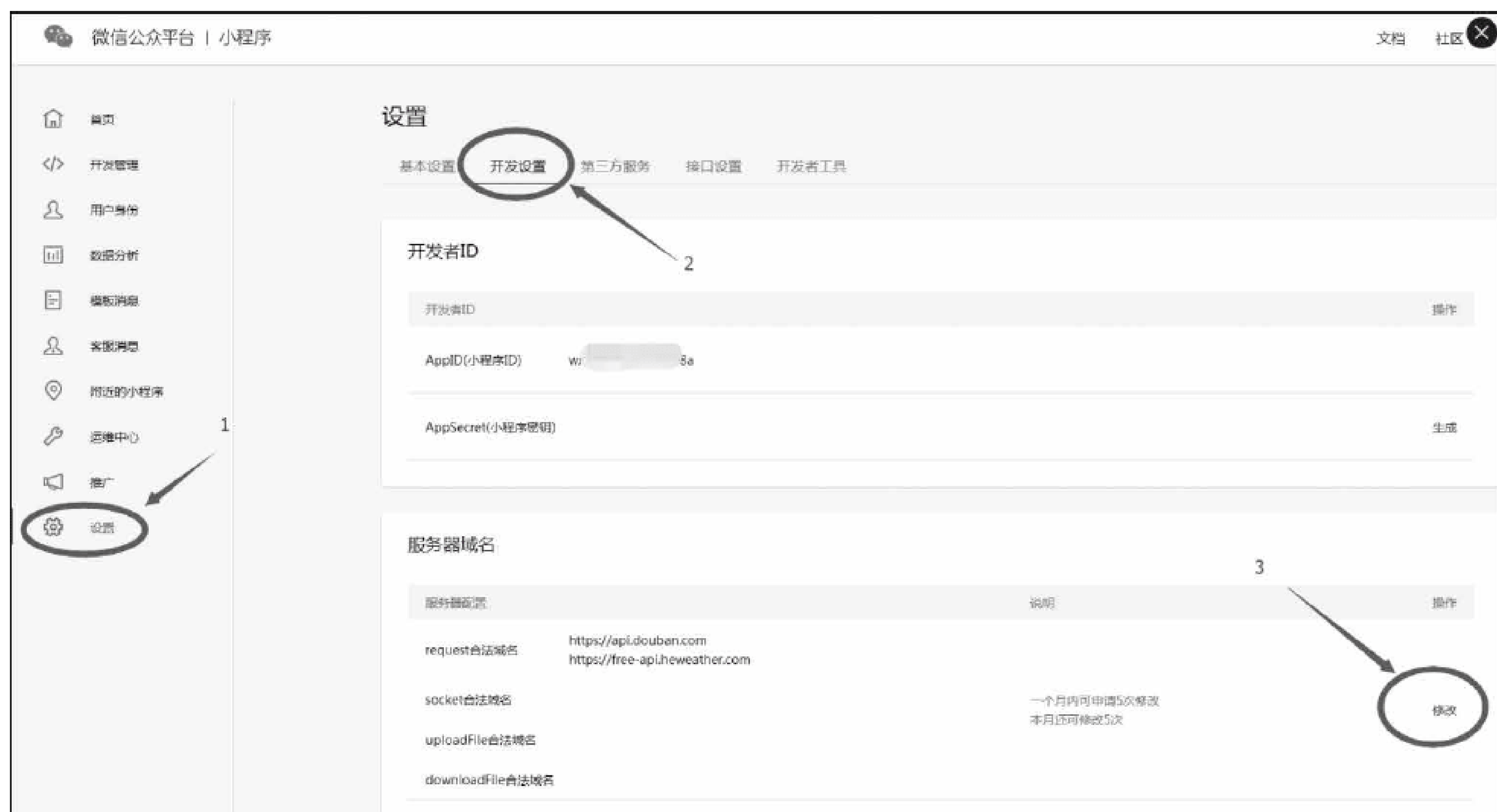


图 A-1 服务器域名配置

开发者可以填入自己或第三方的服务器域名地址，但在配置时需要注意以下几点：

(1) 域名只支持 HTTPS(request、uploadFile、downloadFile) 和 WSS(connectSocket) 协议。

(2) 域名不能使用 IP 地址或 localhost。

(3) 域名必须经过 ICP 备案。

(4) 出于安全考虑，api. weixin. qq. com 不能被配置为服务器域名，相关 API 也不能在小程序内调用，开发者应将 AppSecret 保存到后台服务器中，通过服务器使用 AppSecret 获取

AccessToken,并调用相关API。

(5) 每类接口分别可以配置最多20个域名。

配置完之后再登录小程序开发工具就可以测试小程序与指定服务器域名地址之间的网络通信情况了,注意每个月只可以申请修改5次。

A.1.2　HTTPS证书

需要注意的是,小程序必须使用HTTPS请求,普通的HTTP请求是不能用于正式环境的。判断HTTPS请求的依据是小程序内会对服务器域名使用的HTTPS证书进行校验,如果校验失败,则请求不能成功发起。

因此开发者如果选择自己的服务器需要在服务器上自行安装HTTPS证书,选择第三方服务器则需要确保其HTTPS证书有效。小程序对证书的要求如下:

(1) HTTPS证书必须有效,证书必须被系统信任,部署SSL证书的网站域名必须与证书颁发的域名一致,证书必须在有效期内。

(2) iOS不支持自签名证书。

(3) iOS下证书必须满足苹果App Transport Security (ATS)的要求。

(4) TLS必须支持1.2及以上版本,部分旧Android机型还未支持TLS 1.2,请确保HTTPS服务器的TLS版本支持1.2及以下版本。

(5) 部分CA可能不被操作系统信任(例如Chrome 56/57内核对WoSign、StartCom证书限制),请开发者在选择证书时注意小程序和各系统的相关通告。

由于系统限制,不同平台对于证书要求的严格程度不同。为了保证小程序的兼容性,建议开发者按照最高标准进行证书配置,并使用相关工具检查现有证书是否符合要求。

 A.2　临时服务器部署 

A.2.1　软件部署

若开发者条件受限,可以将个人计算机临时部署为模拟服务器进行开发和测试。小程序对服务器端没有软件和语言的限制条件,用户可以根据自己的实际情况选择Apache、Ngnix、Tomcat等任意一款服务器软件进行安装部署,以及选用PHP、Node.js、J2EE等任意一种语言进行后端开发。

这里以phpStudy 2016套装软件(包含了Apache和PHP)为例,部署步骤如下:

(1) 下载安装包,在本地计算机中双击安装。

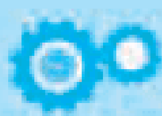
(2) 完成后启动Apache+MySQL服务,启动画面如图A-2所示。

此时模拟服务器就已经启动,打开浏览器在地址栏中输入"http://localhost",如果可以访问成功(如图A-3所示),说明Apache和PHP已经开始工作了。

将页面往下翻到"MySQL数据库连接检测"版块,数据库的初始用户名和密码均为root,输入后单击"MySQL数据库连接检测"按钮进行检测(如图A-4所示)。

此时如果连接成功会出现图A-5的提示,说明MySQL数据库可以正常使用了。

需要注意的是,部分计算机在安装启动时会提示缺少某版本的Visual C++库。使用的PHP版本和运行库的对照关系如下:



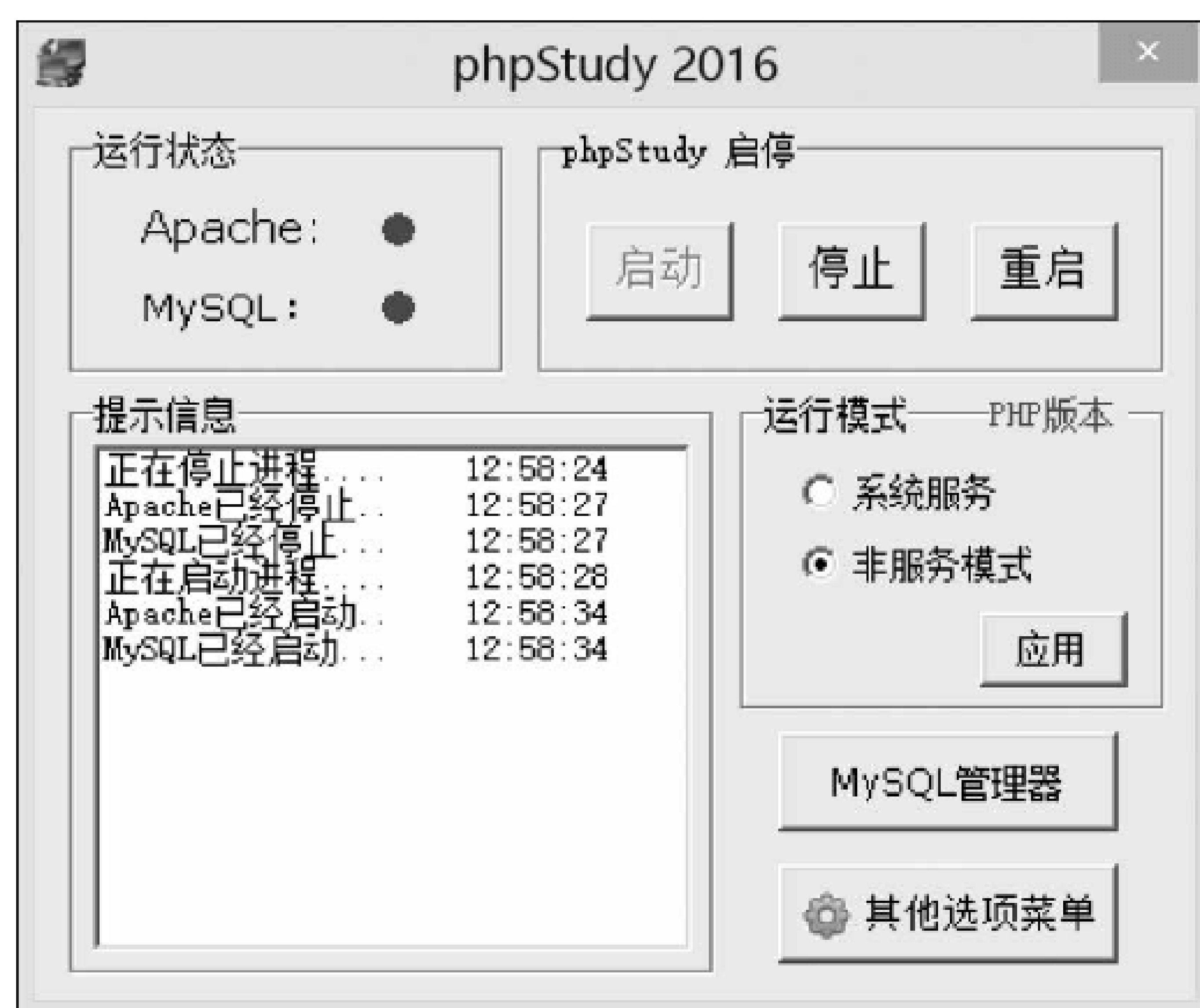


图 A-2　模拟服务器启动状态

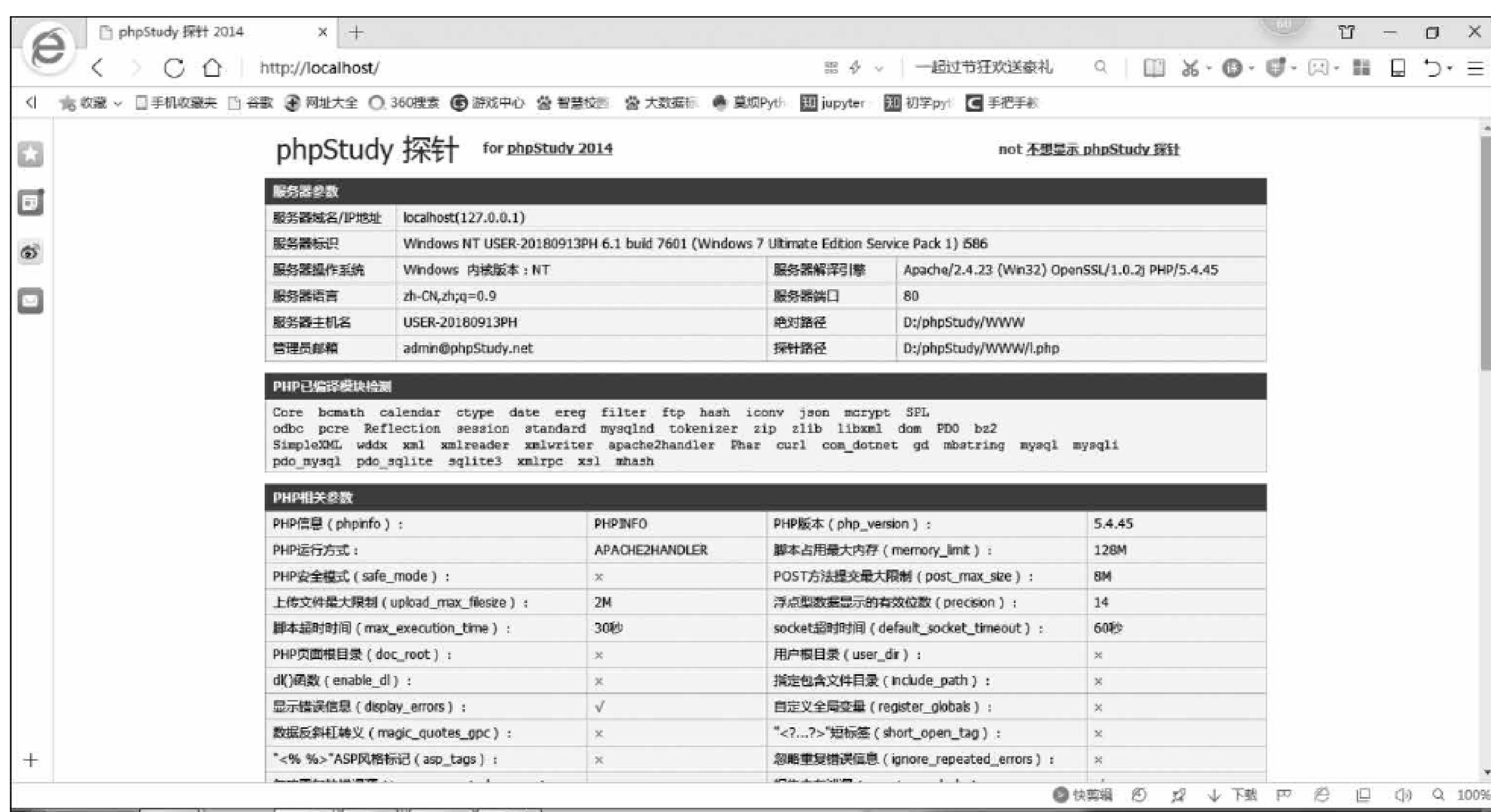


图 A-3　本地页面访问成功



图 A-4　MySQL 数据库连接检测



图 A-5　MySQL 数据库连接正常提示消息

（1）PHP 5.3、PHP 5.4 和 Apache 都是用 VC9 编译，计算机必须安装 VC9 运行库才能运行。

（2）PHP 5.5、PHP 5.6 是用 VC11 编译，如用 PHP 5.5、PHP 5.6 必须安装 VC11 运行库。

（3）PHP 7.0、PHP 7.1 是用 VC14 编译，如用 PHP 7.0、PHP 7.1 及以上版本必须安装 VC14 运行库。

A.2.2　网络请求

服务器安装路径下的 WWW 目录就是根目录，它的网络地址是"http://localhost/"或"http://127.0.0.1/"。开发者可以在根目录下自行创建目录和文件，例如在 miniDemo 中创建了 test.php 文件，那么网络请求地址就是"http://localhost/miniDemo/test.php"。

PHP 文件的返回语句是 echo，例如：

```
1.  <?php
2.      echo '网络请求成功!';
3.  ?>
```

这样小程序将会收到引号里面的文字内容。开发者也可以直接用浏览器访问该地址，能获得同样的文字内容，因此可以在开发之前直接使用浏览器测试 PHP 文件是否正确。

需要注意的是，本地模拟服务器地址只能用于学习或测试阶段，带有无效域名的小程序是无法正式发布上线的。未来在正式服务器域名配置成功后，建议开发者更新网络请求地址并在各平台下进行测试，以确认服务器域名配置正确。

A.2.3　跳过域名校验

如果开发者暂时无法登记有效域名，可以在开发和测试环节暂时跳过域名校验。具体做法是在微信 web 开发者工具中单击右上角的"详情"按钮，然后勾选"不校验合法域名、web-view（业务域名）、TLS 版本以及 HTTPS 证书"选项，如图 A-6 所示。

此时，在开发者工具中运行或开启手机调试模式时都不会进行服务器域名的校验。



图 A-6　跳过域名校验设置

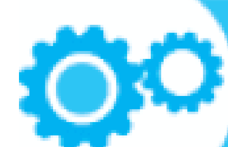


附录B APPENDIX B



可视化数据库搭建

为了方便对 MySQL 数据库的操作,这里推荐一款可视化数据库工具软件 Navicat for MySQL。这款软件是一套专为 MySQL 设计的高性能数据库管理及开发工具,支持绝大部分 MySQL 最新版本的功能。



B.1 软件部署



这里假设已经安装完成了 phpStudy 2016 套装软件并已经启动了 Apache 与 MySQL 服务(安装和启动方式见“附录 A”),Navicat for MySQL 工具的部署步骤如下:

(1) 下载安装包,在本地计算机中双击安装。

(2) 完成后启动 Navicat。

(3) 单击左上角的“连接”图标(如图 B-1 所示),在弹出的对话框中分别填入自定义连接名、主机地址、端口、用户名和密码,并单击“确定”按钮完成连接(如图 B-2 所示)。

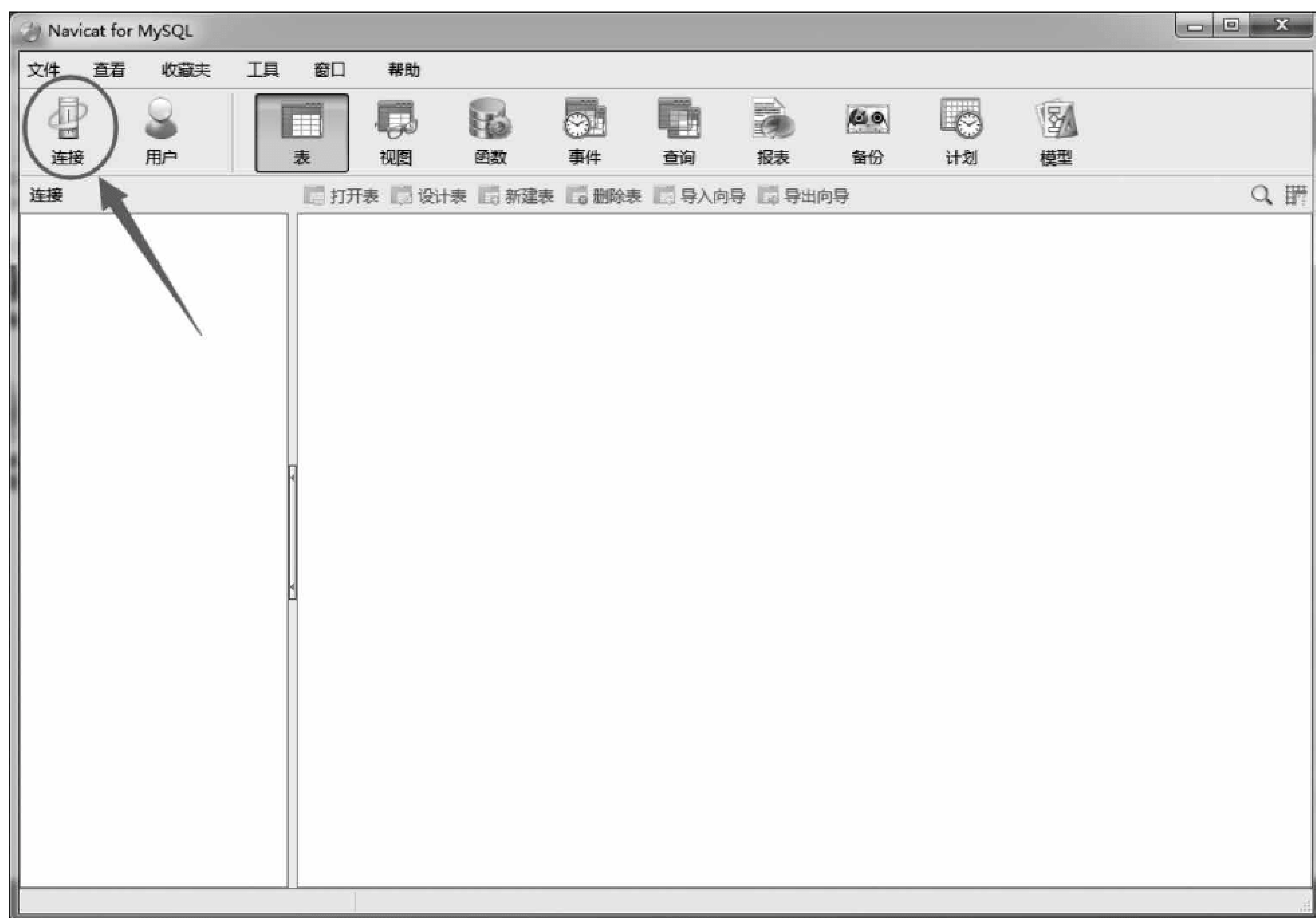


图 B-1　创建数据库连接



图 B-2 填入数据库连接信息

注意，如果是使用 phpStudy 2016 套件进行安装的，初始用户名和密码均为 root。完成后会在左侧列表中出现连接名信息，如图 B-3 所示。

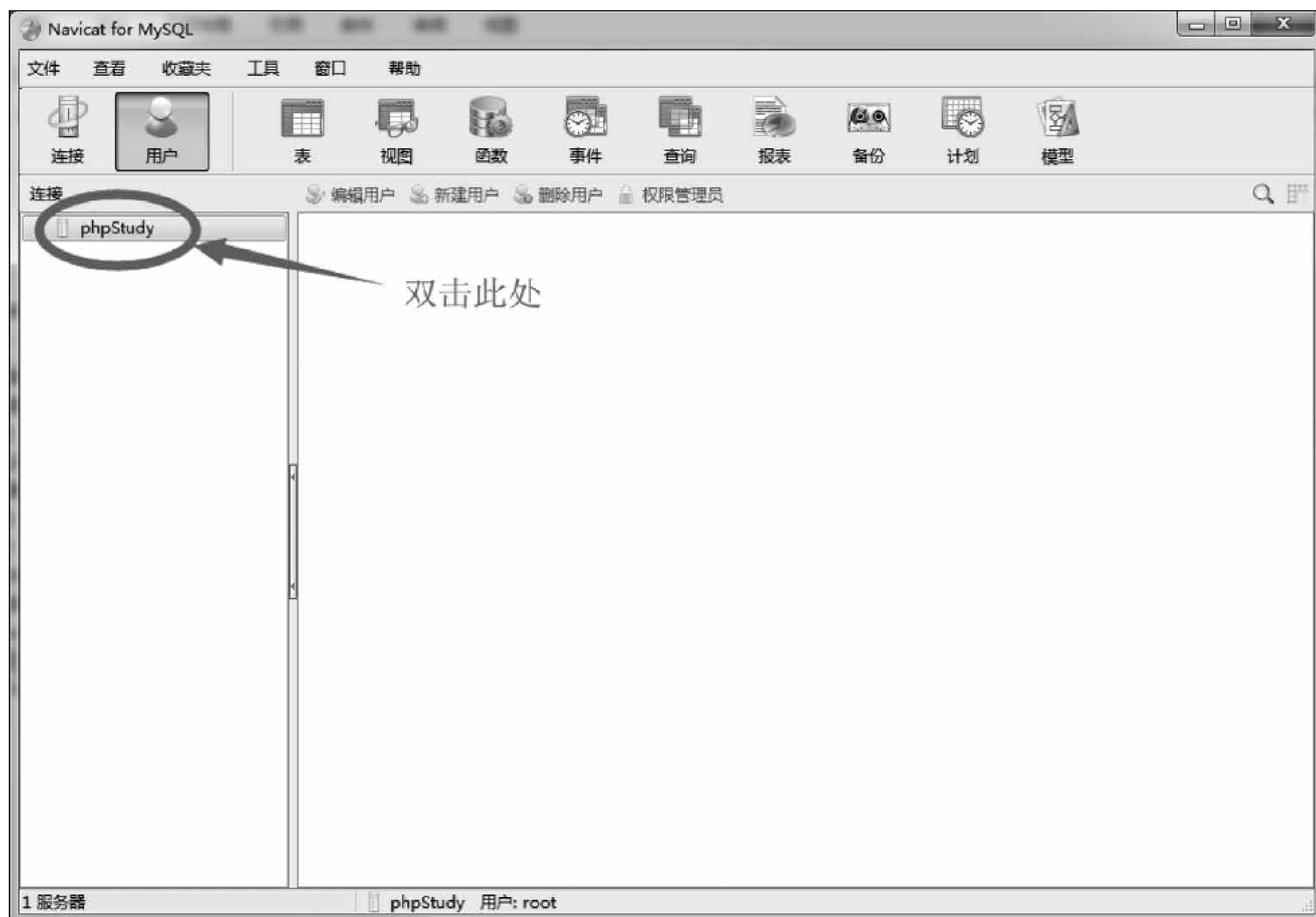
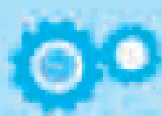


图 B-3 数据库连接列表

双击连接名后可看到已经存在的数据库列表，可以新建自定义名称的数据库。





B.2 创建数据库



在连接创建并激活后，可以右击该连接，选择“新建数据库”命令，如图 B-4 所示。

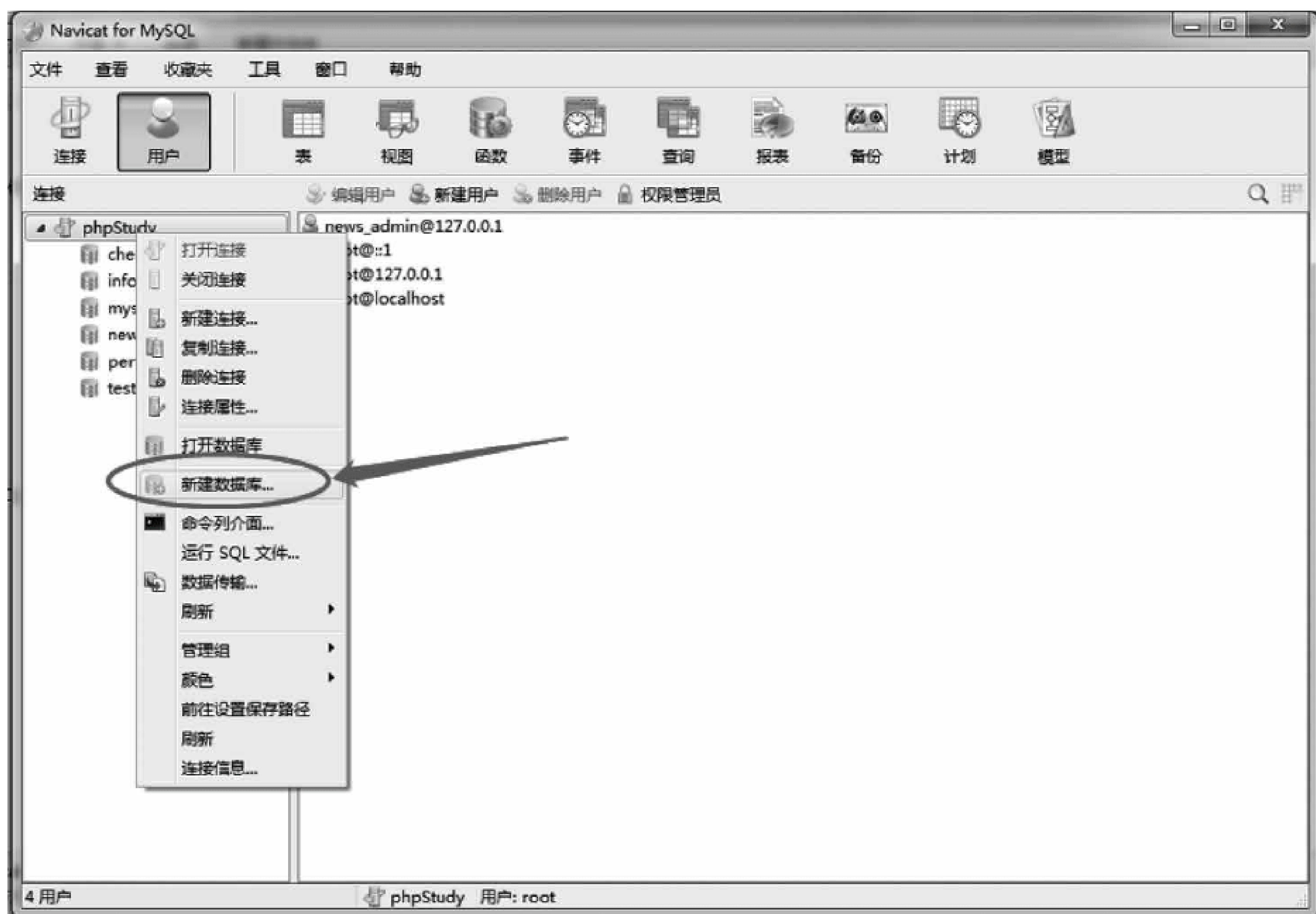


图 B-4 选择“新建数据库”命令

在弹出的“新建数据库”对话框中填入自定义数据库名称 news，并使用下拉列表选择字符集 utf8mb4--UTF-8 Unicode 和排序规则 utf8mb4_general_ci，如图 B-5 所示。

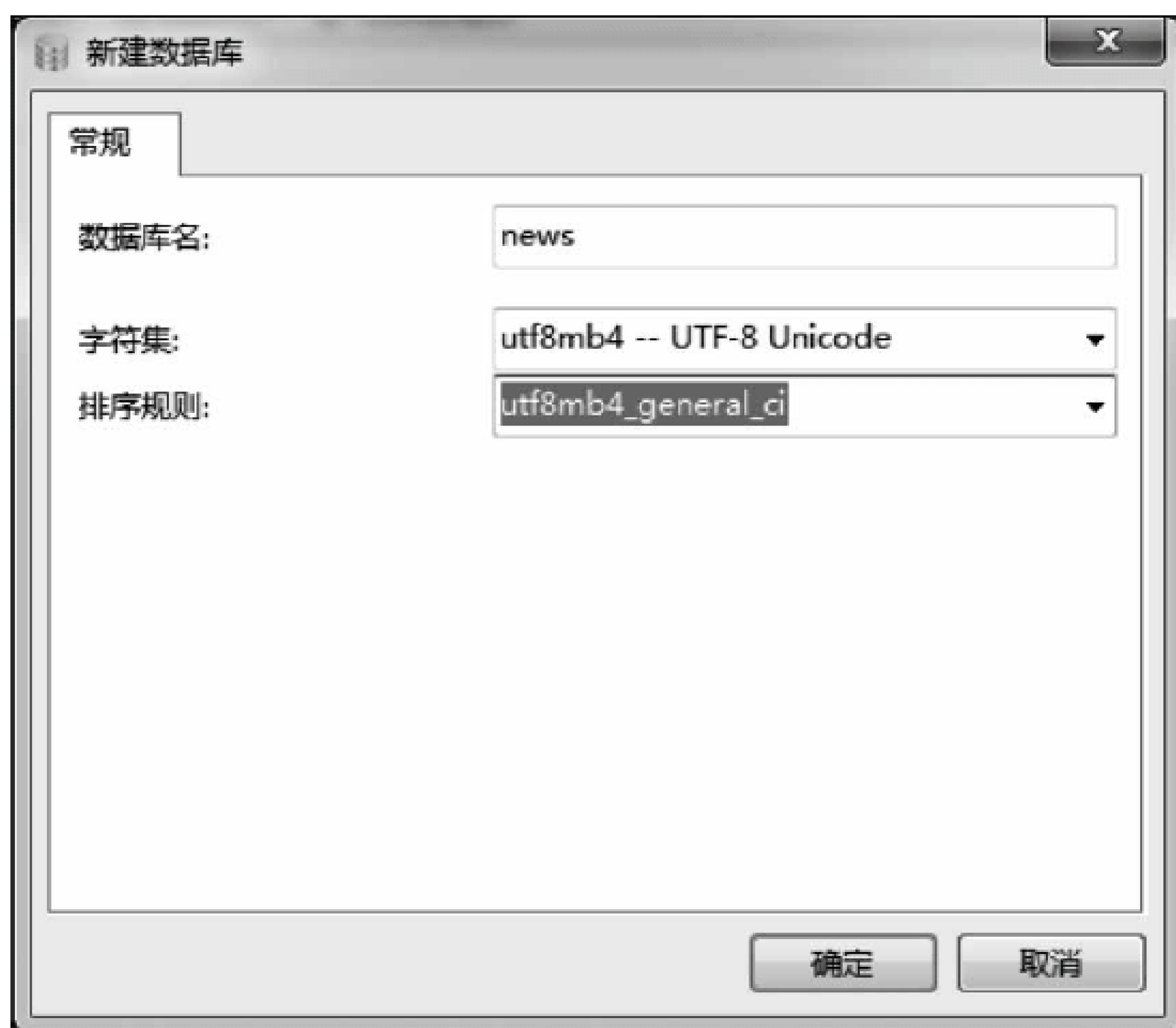


图 B-5 “新建数据库”对话框

单击“确定”按钮后完成数据库的创建，双击进入数据库，并单击“表”→“新建表”，如图 A-6 所示。

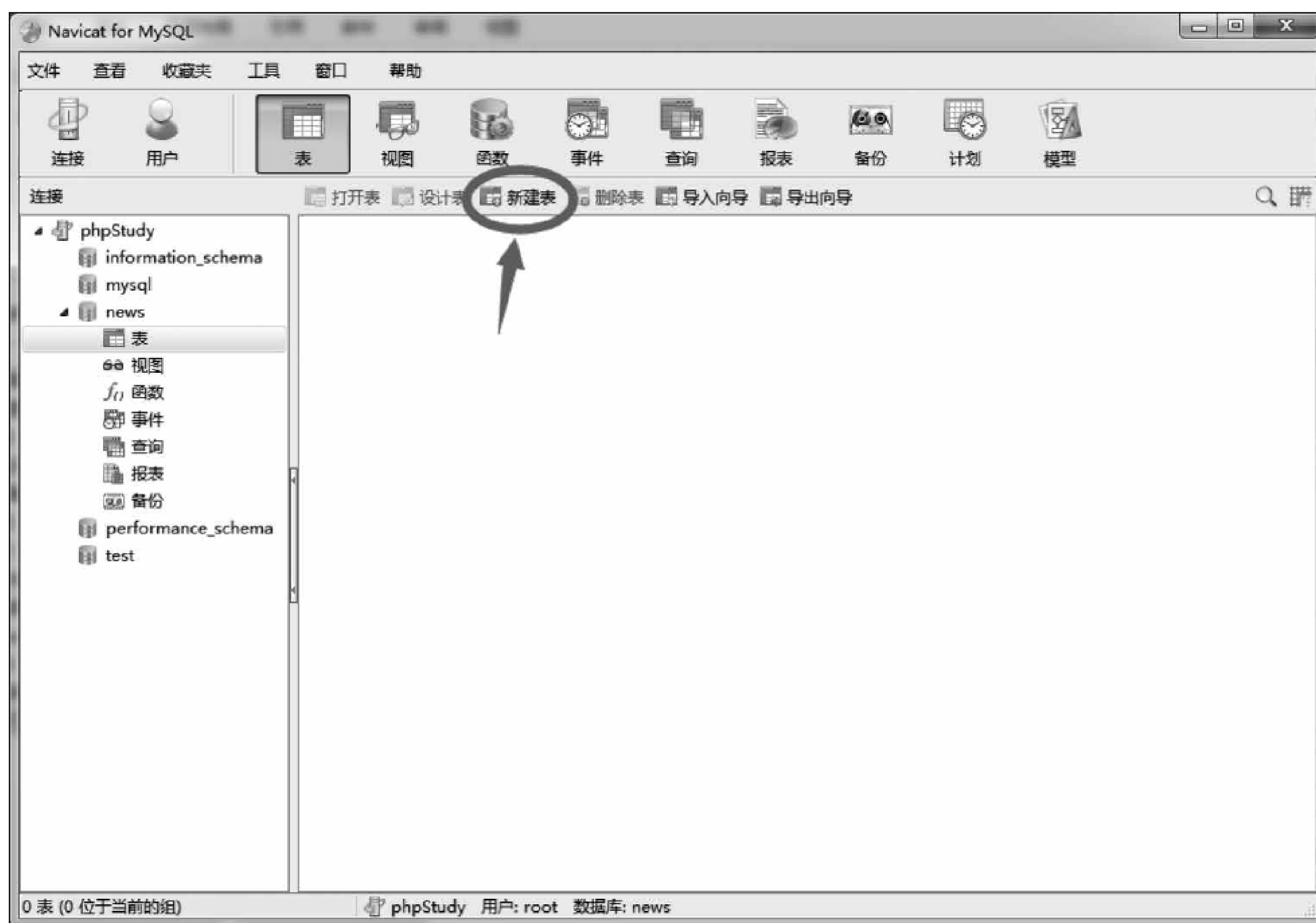


图 B-6　新建数据表

然后根据实际需要录入表的字段名称、类型、长度等信息（如图 B-7 所示），保存后录入数据表名称，例如 campus_news。

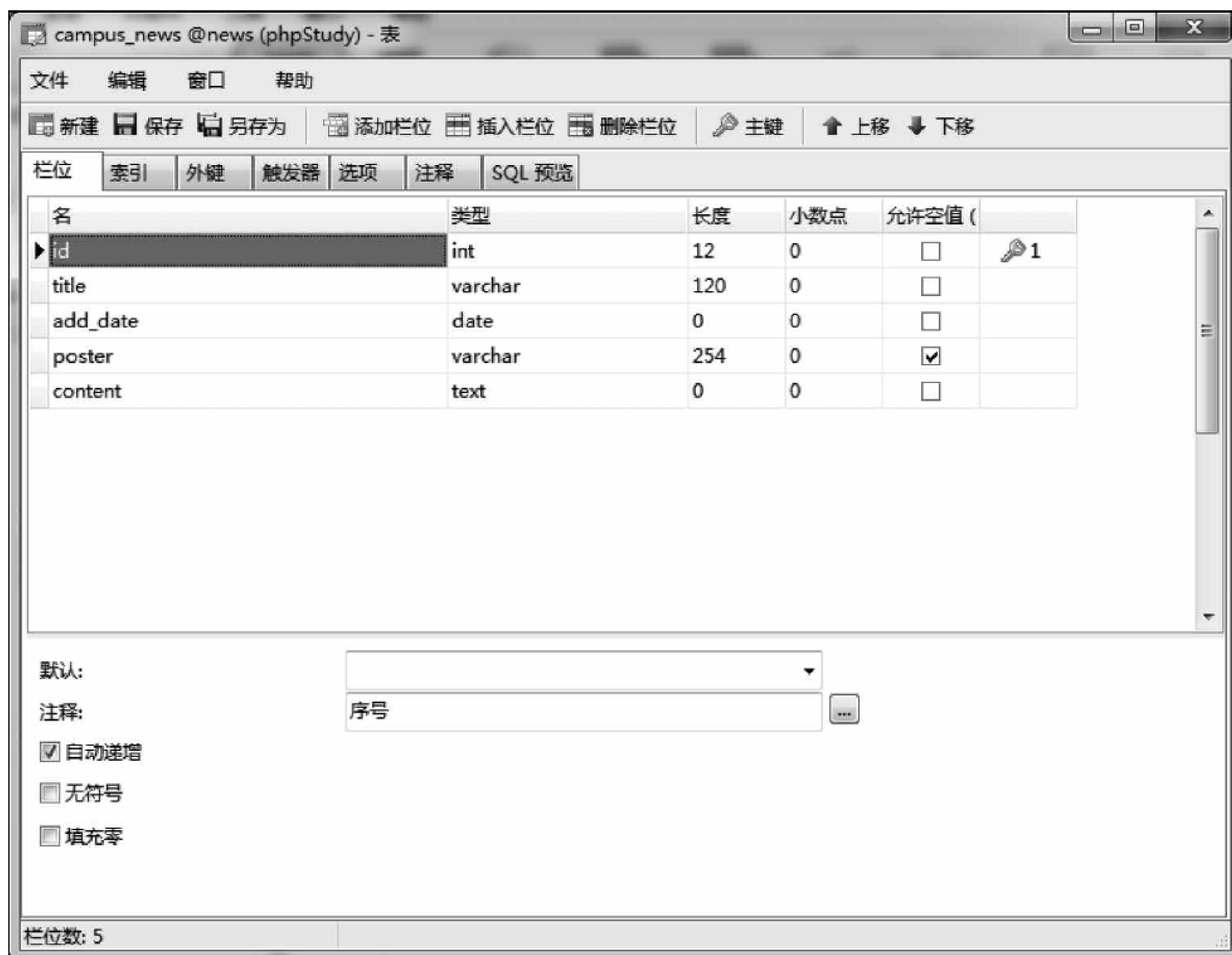
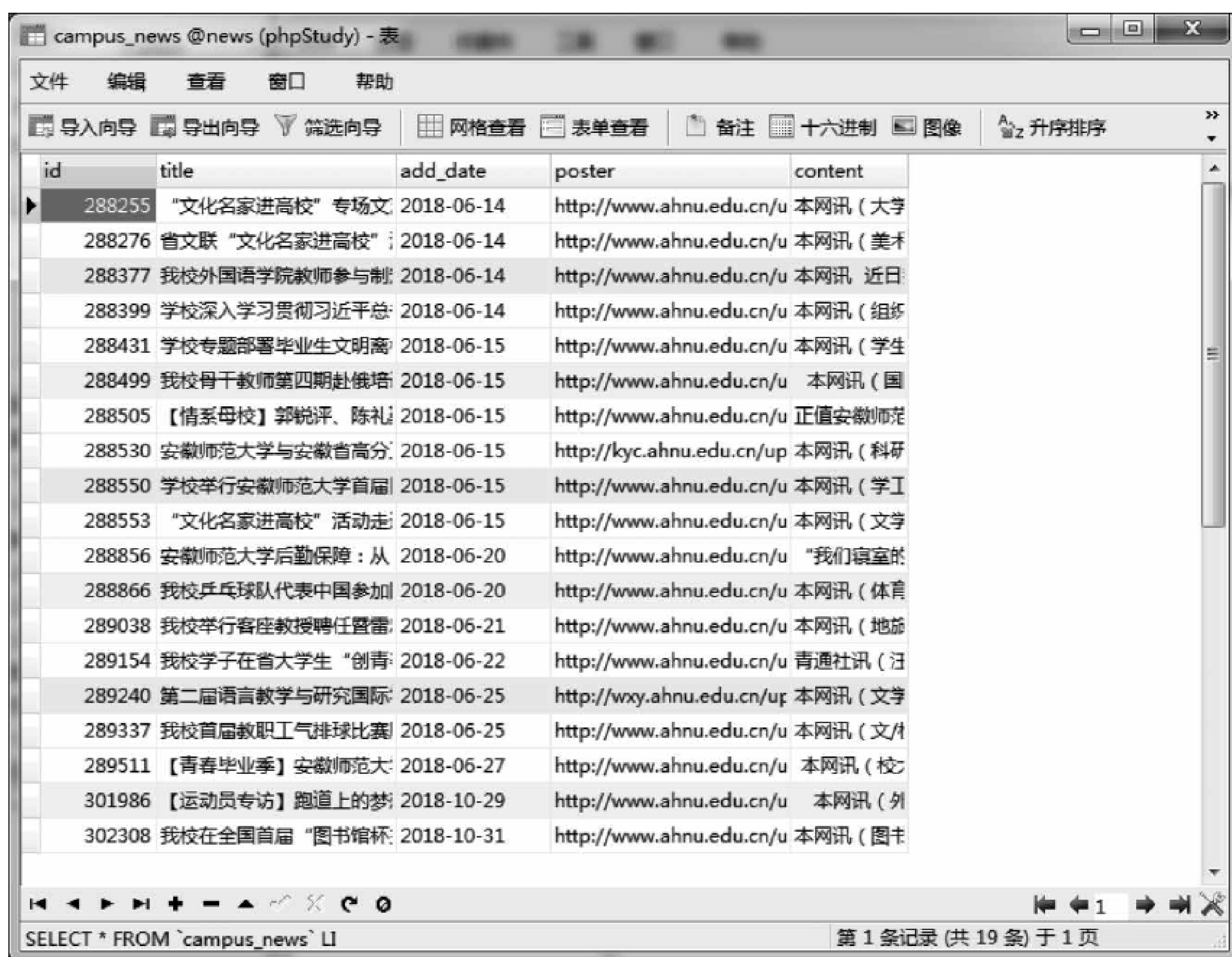


图 B-7　创建数据表字段信息

双击打开数据表，并录入若干条数据，如图 B-8 所示。



id	title	add_date	poster	content
288255	"文化名家进高校"专场文	2018-06-14	http://www.ahnu.edu.cn/u	本网讯（大学
288276	省文联"文化名家进高校"	2018-06-14	http://www.ahnu.edu.cn/u	本网讯（美术
288377	我校外国语学院教师参与制	2018-06-14	http://www.ahnu.edu.cn/u	本网讯 近日
288399	学校深入学习贯彻习近平总	2018-06-14	http://www.ahnu.edu.cn/u	本网讯（组织
288431	学校专题部署毕业生文明离	2018-06-15	http://www.ahnu.edu.cn/u	本网讯（学生
288499	我校骨干教师第四期赴俄培	2018-06-15	http://www.ahnu.edu.cn/u	本网讯（国
288505	【情系母校】郭锐评、陈礼	2018-06-15	http://www.ahnu.edu.cn/u	正值安徽师范
288530	安徽师范大学与安徽省高分	2018-06-15	http://kyc.ahnu.edu.cn/up	本网讯（科研
288550	学校举行安徽师范大学首届	2018-06-15	http://www.ahnu.edu.cn/u	本网讯（学工
288553	"文化名家进高校"活动走	2018-06-15	http://www.ahnu.edu.cn/u	本网讯（文学
288856	安徽师范大学后勤保障：从	2018-06-20	http://www.ahnu.edu.cn/u	"我们寝室的
288866	我校乒乓球队代表中国参加	2018-06-20	http://www.ahnu.edu.cn/u	本网讯（体育
289038	我校举行客座教授聘任暨	2018-06-21	http://www.ahnu.edu.cn/u	本网讯（地旅
289154	我校学子在省大学生"创青	2018-06-22	http://www.ahnu.edu.cn/u	青通社讯（注
289240	第二届语言教学与研究国际	2018-06-25	http://wxy.ahnu.edu.cn/up	本网讯（文学
289337	我校首届教职工气排球比赛	2018-06-25	http://www.ahnu.edu.cn/u	本网讯（文/
289511	【青春毕业季】安徽师范大	2018-06-27	http://www.ahnu.edu.cn/u	本网讯（校
301986	【运动员专访】跑道上的梦	2018-10-29	http://www.ahnu.edu.cn/u	本网讯（外
302308	我校在全国首届"图书馆杯	2018-10-31	http://www.ahnu.edu.cn/u	本网讯（图书

图 B-8 在数据表中录入数据

同一个数据库允许创建不同名称的多张数据表。

 B.3 分配用户权限 

一般来说尽量不要使用 root 用户进行后端的接口制作，以免带来安全隐患，可以创建一个独立用户，并赋予它相关数据库的权限。

首先单击顶部的"用户"图标，如图 B-9 所示。

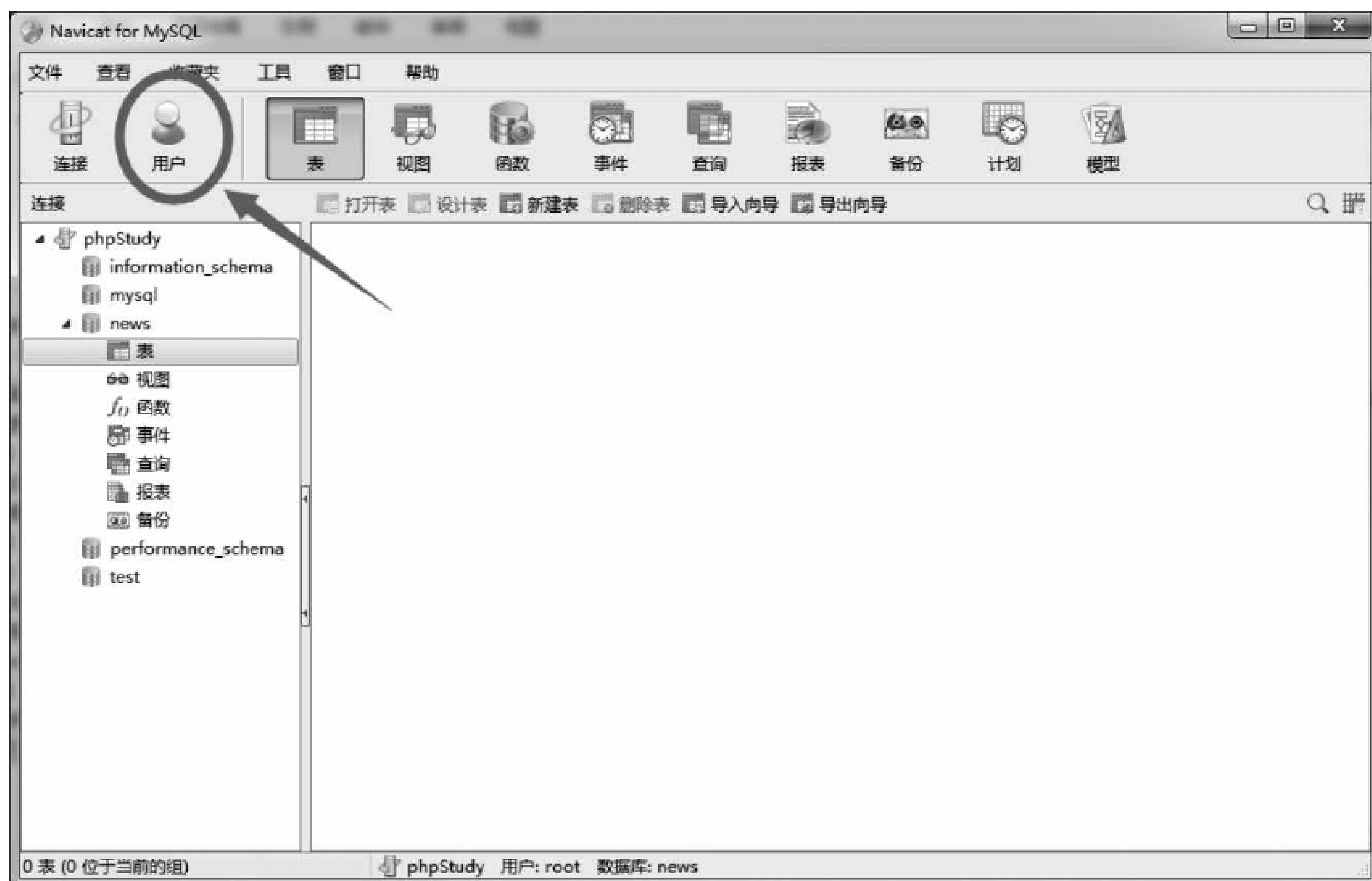


图 B-9 单击"用户"图标

然后单击“新建用户”按钮，切换到用户管理面板，如图 B-10 和图 B-11 所示。

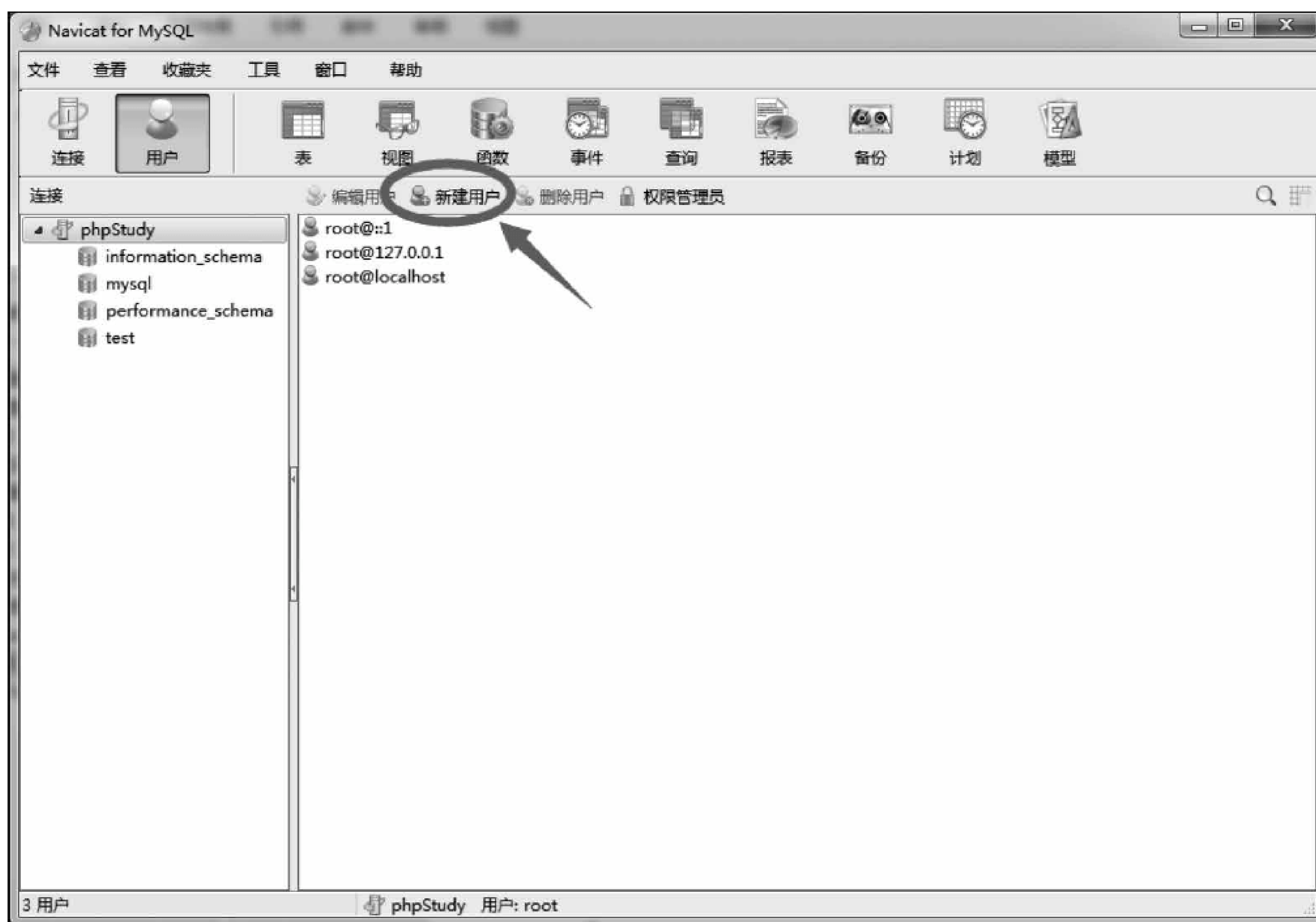


图 B-10 单击“新建用户”按钮

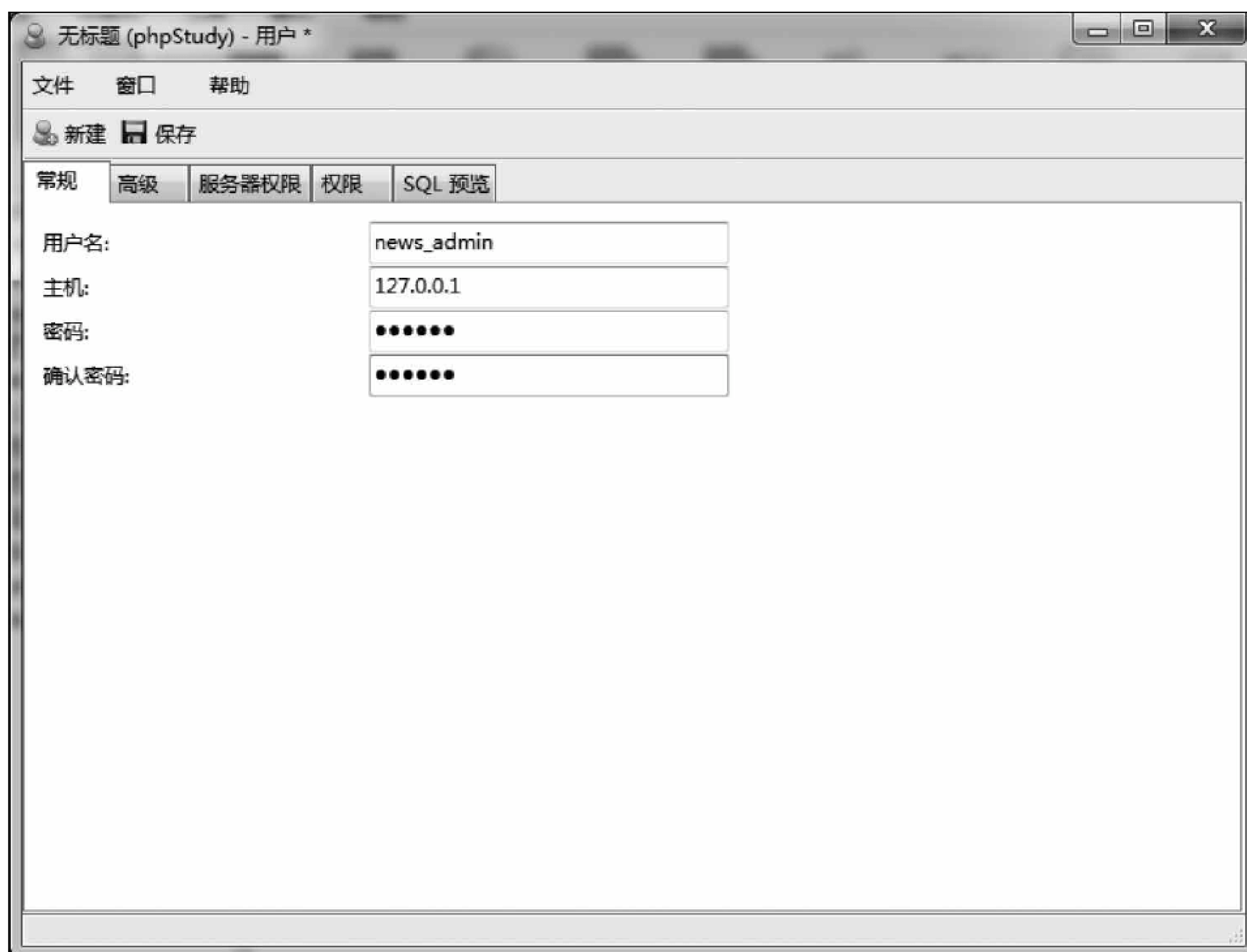


图 B-11 切换到用户管理面板

接着单击“添加权限”按钮，如图 B-12 所示。

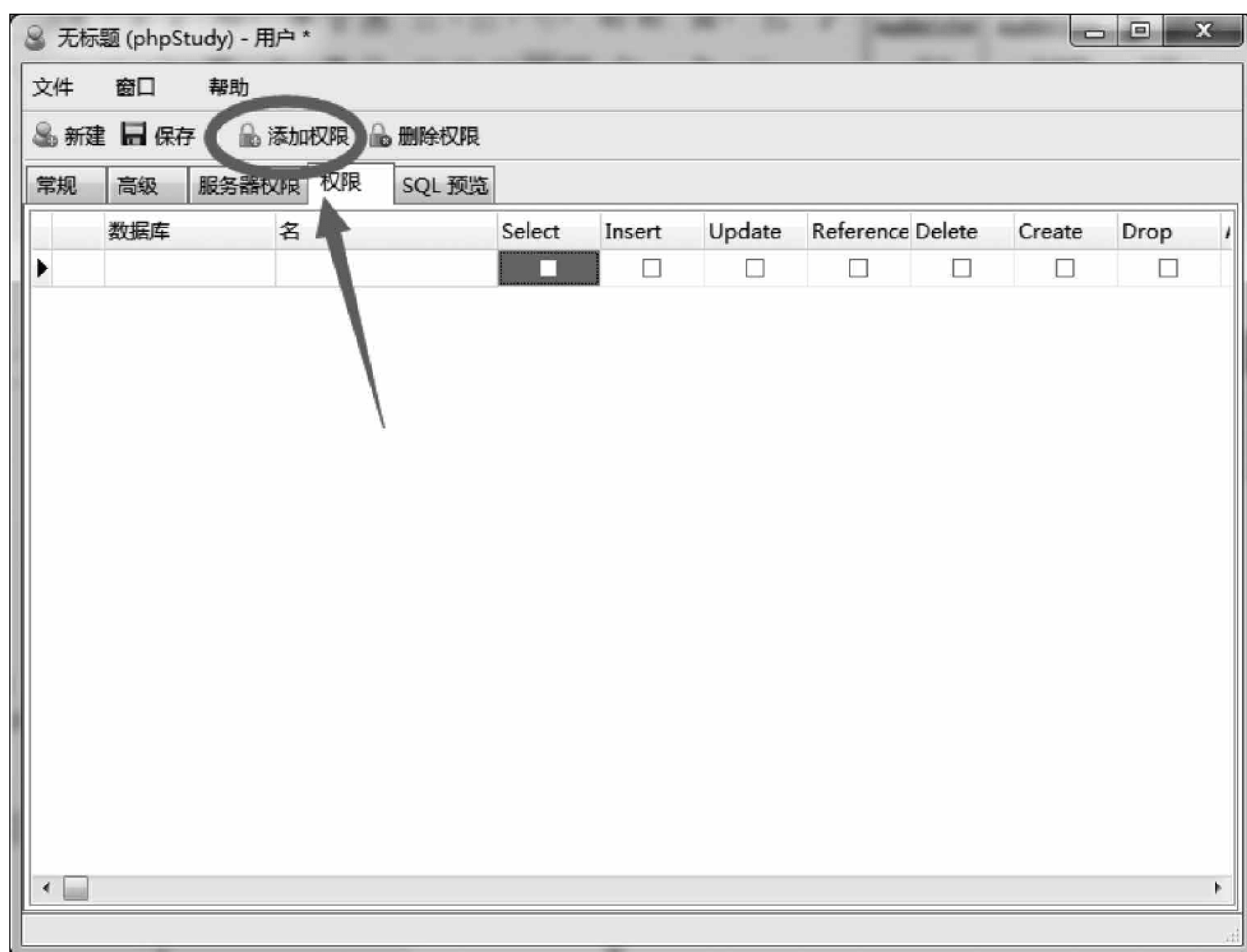


图 B-12　单击“添加权限”按钮

在弹出的“添加权限”对话框中勾选需要赋予权限的数据库 news 和查询权限 Select（如图 B-13 所示），然后单击“确定”按钮完成。

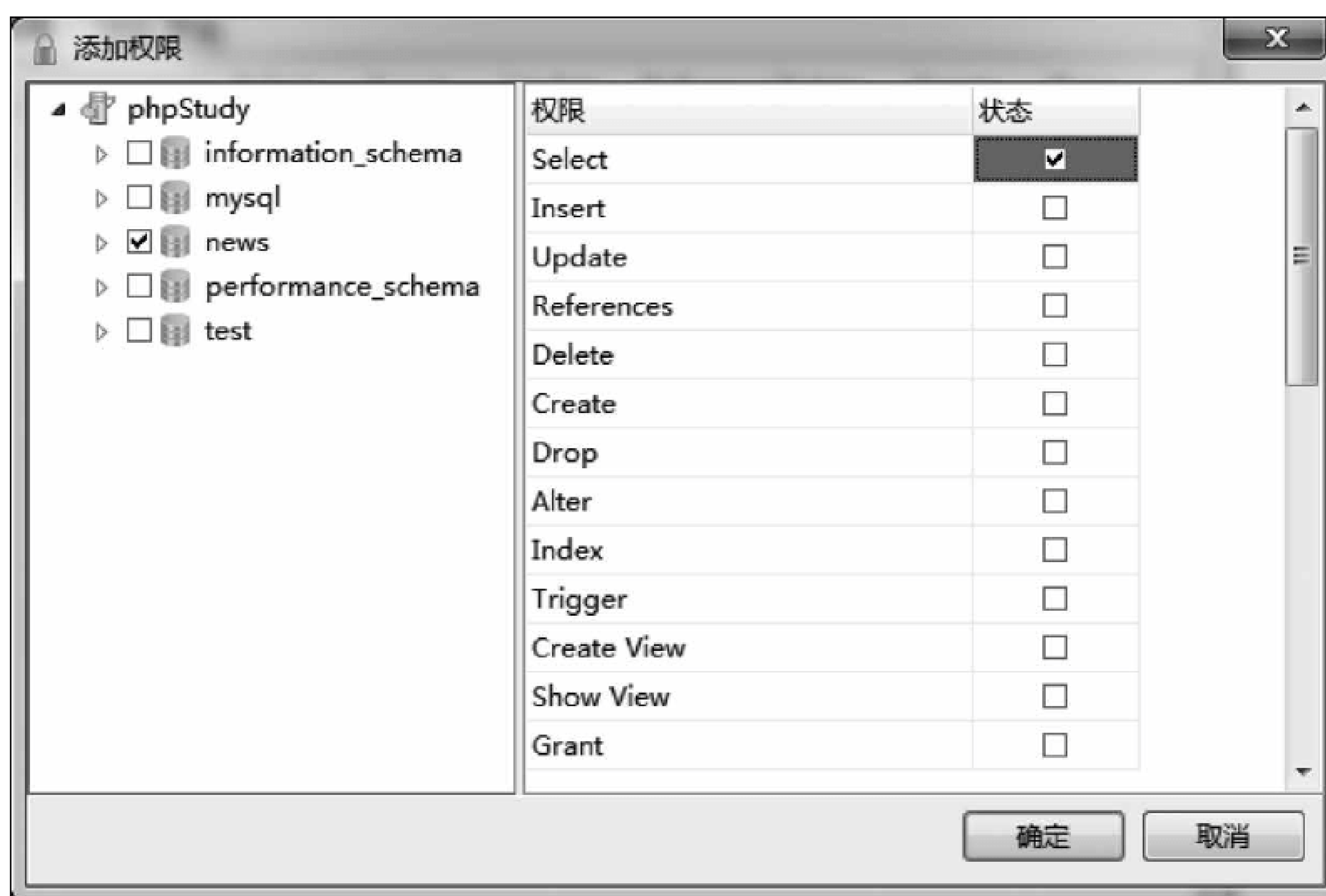


图 B-13　“添加权限”对话框

然后单击“保存”按钮完成权限分配（如图 B-14 所示），这样就可以关闭了。

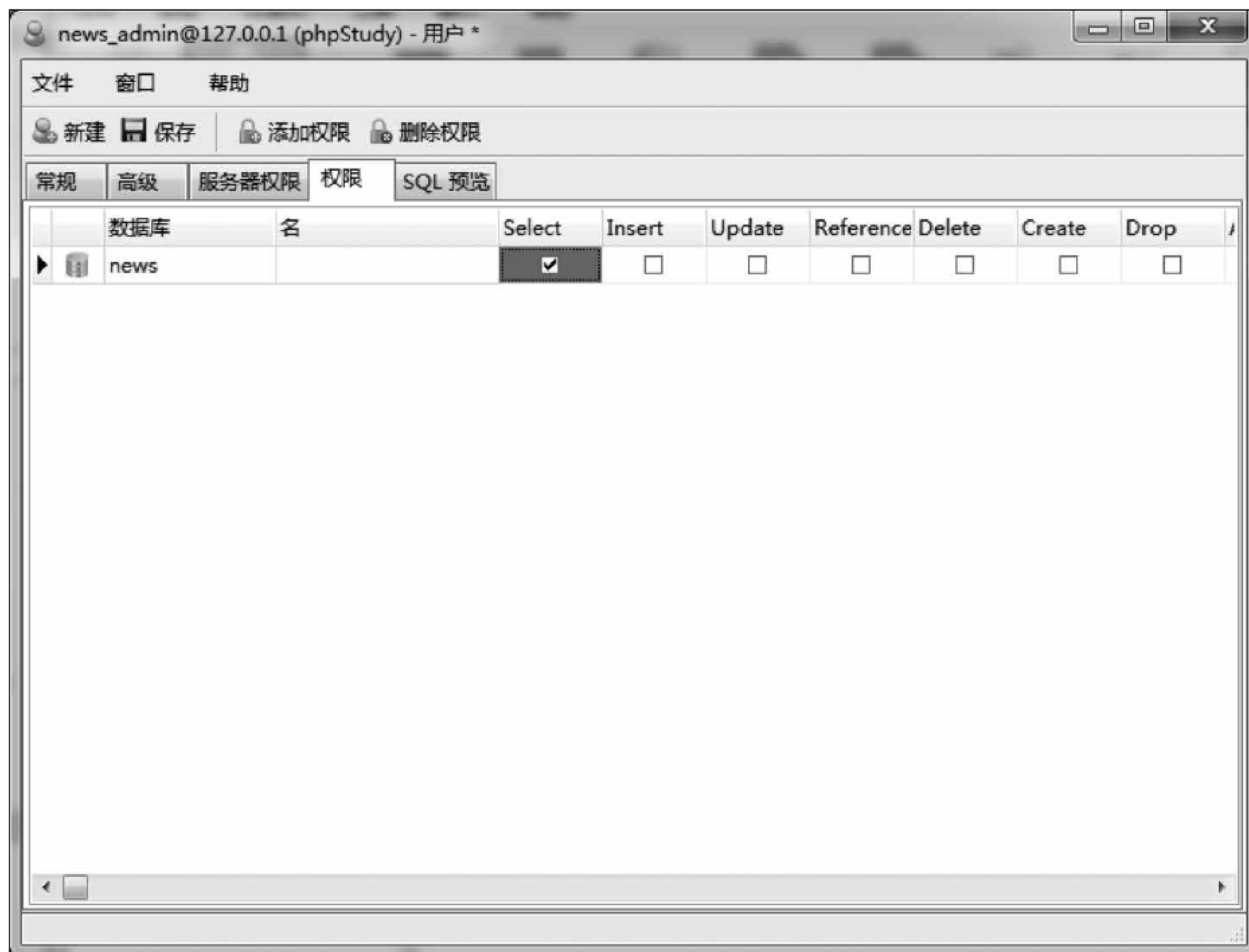


图 B-14 保存后完成权限分配



附录C APPENDIX C



后端框架搭建

小程序允许对接任意语言开发的后端接口，例如 PHP、Node. js、JavaEE 等。用户可以根据实际擅长情况任意选择一款进行后端开发和接口的制作。

对于初学者而言，PHP+MySQL 相对来说比较容易上手。因此这里以免费开源 PHP 框架 ThinkPHP 3. 2. 4 为例，说明如何在本地服务器进行后端框架的部署和接口制作。



C.1 后端框架部署

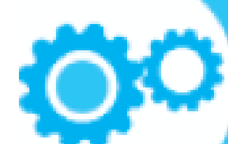


首先将提供的开源框架包 thinkphp_3. 2. 4 整个复制、粘贴到 phpStudy 的 WWW 目录下，并将其重命名成自定义名称，例如 myNews。

在确保 Apache 服务已经启动的情况下，使用浏览器访问“http://127. 0. 0. 1/myNews”或“http://localhost/myNews”来安装该框架。当看到欢迎使用界面（如图 C-1 所示）时表示安装成功。



图 C-1　欢迎使用界面



C.2 数据库对接配置



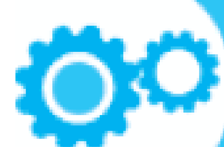
在确保 Apache 和 MySQL 服务已经启动的情况下，找到 myNews 文件夹下的公共配置文件 config. php（路径地址为 Application/Common/Conf/config. php），用文本编辑工具打开进行数据库对接配置。

参考代码如下：

```
1.  <?php
2.  return array(
3.    //'配置项'=>'配置值'
4.    'DB_TYPE'=>'mysql',                //数据库类型
```

```
5.     'DB_HOST' => '127.0.0.1',                 //服务器地址
6.     'DB_NAME' => 'news',                      //数据库名称
7.     'DB_USER' => 'news_admin',                //数据库用户名
8.     'DB_PWD' => '123456',                     //数据库密码
9.     'DB_PORT' => 3306,                        //端口号
10.    'DB_PARAMS' => array(),                   //数据库参数
11.    'DB_CHARSET' => 'utf8',                   //数据库字符集
12.    'DB_DEBUG' => TRUE,                       //数据库允许 debug
13.    'MODULE_ALLOW_LIST' => array('Home'),     //允许访问的模块
14.    'DEFAULT_MODULE' => 'Home',               //默认模块
15.    'DEFAULT_CONTROLLER' => 'Index',          //默认控制器名称
16. );
```

请根据数据库的实际情况修改对应的服务器地址、数据库名称、用户名、密码、端口号等内容，使其能够与服务器进行对接。

 C.3　接口制作示例 

然后找到框架文件夹 myNews 中的控制器文件 IndexController.class.php（路径地址为 Application/Home/Controller/IndexController.class.php），用文本编辑器打开进行代码编写。

例如查询数据库中一张名为 student 的表中的全部数据：

```
1.  <?php
2.  namespace Home\Controller;
3.  use Think\Controller;
4.
5.  class IndexController extends Controller
6.  {
7.     public function getStuInfo(){
8.         $m = M();                                      //创建查询模型
9.         $rs = $m->query('select * from student');      //执行 SQL 语句并获得查询结果
10.        echo json_encode($rs);                         //把 JSON 格式数据发出去
11.        }
12. }
```

此时接口就制作完成了，其中函数名 getStuInfo 为自定义名称，开发者可以自行更改。

接口地址的格式如下：

```
http://服务器地址/框架名/控制器名称/函数名[/参数 1/值 1…/参数 N/值 N]
```

因此本次示例的接口地址为"http://127.0.0.1/myNews/Index/getStuInfo"可以直接使用浏览器进行测试访问，并且字母大小写不敏感。

接下来示范一下带参数查询。

例如查询数据库中一张名为 student 的表中指定某学号 id 的学生数据：

```
1.  <?php
2.  namespace Home\Controller;
3.  use Think\Controller;
4.
5.  class IndexController extends Controller
6.  {
```

```
7.      public function getStuById(){
8.          $id = I('id');                          //获取参数 id 的值
9.          $m = M();                               //创建查询模型
10.         $rs = $m->query('select * from student where id ='.$id);  //获得查询结果
11.         echo json_encode($rs);                  //把 JOSN 格式数据发出去
12.     }
13. }
```

其中函数名 getStuById 为自定义名称，开发者可以自行更改。

此时接口地址为“http://127.0.0.1/myNews/Index/getStuById”，如果想使用浏览器进行测试访问，带参数示范的完整地址为“http://127.0.0.1/myNews/Index/getStuById/id/123”。

注意：对于接口制作所使用的数据库、后端语言和框架均没有固定要求，这里仅是其中一种示例，开发者完全可以根据自己所擅长的领域自由选择。




附录D APPENDIX D

个人开发者服务类目以及小程序场景值、小程序预定颜色

个人开发者服务类目如表 D-1 所示。

表 D-1　个人开发者服务类目

一级分类	二级分类	三级分类
快递业与邮政	快递、物流	寄件/收件
		查件
	邮政	—
	装卸/搬运	—
教育	教育信息服务	—
	特殊人群教育	—
	婴幼儿教育	—
	在线教育	—
	教育装备	—
出行与交通	代驾	—
生活服务	票务	—
	生活缴费	—
	家政	—
	外送	—
	环保回收/废品回收	—
	摄影/扩印	—
	婚庆服务	—
餐饮	点评与推荐	—
	菜谱	—
	餐厅排队	—
旅游	旅游攻略	—
	出境 Wi-Fi	—
工具	记账	—
	日历	—
	天气	—
	办公	—
	字典	—
	图片/音频/视频	—

续表

一级分类	二级分类	三级分类
工具	计算类	—
	报价/比价	—
	信息查询	
	效率	—
	预约	—
	健康管理	—
	企业管理	—
商业服务	法律服务	法律咨询
		在线法律服务
	会展服务	—
	一般财务服务	—
	农林牧渔	—
体育	体育培训	—
	在线健身	—

注意：面向个人开发者开放的服务类目会随着相关政策、法律法规以及平台规定的变化而变化，请开发者以提交时开放的类目为准，本文档仅供参考。

小程序场景值如表 D-2 所示。

表 D-2　小程序场景值

场景值 ID	说　明
1001	发现栏小程序主入口，“最近使用”列表(从基础库 2.2.4 版本起将包含“我的小程序”列表)
1005	顶部搜索框的搜索结果页
1006	发现栏小程序主入口，搜索框的搜索结果页
1007	单人聊天会话中的小程序消息卡片
1008	群聊会话中的小程序消息卡片
1011	扫描二维码
1012	长按图片识别二维码
1013	手机相册选取二维码
1014	小程序模板消息
1017	前往体验版的入口页
1019	微信钱包
1020	公众号 profile 页相关小程序列表
1022	聊天顶部置顶小程序入口
1023	安卓系统桌面图标
1024	小程序 profile 页
1025	扫描一维码
1026	附近小程序列表
1027	顶部搜索框的搜索结果页，“使用过的小程序”列表
1028	我的卡包
1029	卡券详情页
1030	自动化测试下打开小程序
1031	长按图片识别一维码

续表

场景值 ID	说　明
1032	手机相册选取一维码
1034	微信支付完成页
1035	公众号自定义菜单
1036	App 分享消息卡片
1037	小程序打开小程序
1038	从另一个小程序返回
1039	摇电视
1042	添加好友搜索框的搜索结果页
1043	公众号模板消息
1044	带 shareTicket 的小程序消息卡片(详情)
1045	朋友圈广告
1046	朋友圈广告详情页
1047	扫描小程序码
1048	长按图片识别小程序码
1049	手机相册选取小程序码
1052	卡券的适用门店列表
1053	搜一搜的结果页
1054	顶部搜索框的小程序快捷入口
1056	音乐播放器菜单
1057	钱包中的银行卡详情页
1058	公众号文章
1059	体验版小程序绑定邀请页
1064	微信连 Wi-Fi 状态栏
1067	公众号文章广告
1068	附近小程序列表广告
1069	移动应用
1071	钱包中的银行卡列表页
1072	二维码收款页面
1073	客服消息列表下发的小程序消息卡片
1074	公众号会话下发的小程序消息卡片
1077	摇周边
1078	连 Wi-Fi 成功页
1079	微信游戏中心
1081	客服消息下发的文字链
1082	公众号会话下发的文字链
1084	朋友圈广告原生页
1089	微信聊天主界面下拉,“最近使用”栏(从基础库 2.2.4 版本起将包含“我的小程序”栏)
1090	长按小程序右上角菜单唤出最近使用历史
1091	公众号文章商品卡片
1092	城市服务入口
1095	小程序广告组件
1096	聊天记录
1097	微信支付签约页

续表

场景值 ID	说　明
1099	页面内嵌插件
1102	公众号 profile 页服务预览
1103	发现栏小程序主入口,“我的小程序”列表(从基础库 2.2.4 版本起废弃)
1104	微信聊天主界面下拉,“我的小程序”栏(从基础库 2.2.4 版本起废弃)

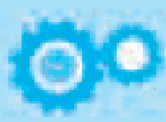
小程序目前预定颜色有 148 个,颜色名称大小写不敏感,如表 D-3 所示。

表 D-3　小程序预定颜色

颜色名称	RGB 十六进制	RGB 十进制	中文名
AliceBlue	#F0F8FF	240,248,255	爱丽丝蓝
AntiqueWhite	#FAEBD7	250,235,215	古董白
Aqua	#00FFFF	0,255,255	青　色
Aquamarine	#7FFFD4	127,255,212	碧　绿
Azure	#F0FFFF	240,255,255	青白色
Beige	#F5F5DC	245,245,220	米　色
Bisque	#FFE4C4	255,228,196	陶坯黄
Black	#000000	0,0,0	黑　色
BlanchedAlmond	#FFEBCD	255,235,205	杏仁白
Blue	#0000FF	0,0,255	蓝　色
BlueViolet	#8A2BE2	138,43,226	蓝紫色
Brown	#A52A2A	165,42,42	褐　色
BurlyWood	#DEB887	222,184,135	硬木褐
CadetBlue	#5F9EA0	95,158,160	军服蓝
Chartreuse	#7FFF00	127,255,0	查特酒绿
Chocolate	#D2691E	210,105,30	巧克力色
Coral	#FF7F50	255,127,80	珊瑚红
CornflowerBlue	#6495ED	100,149,237	矢车菊蓝
Cornsilk	#FFF8DC	255,248,220	玉米穗黄
Crimson	#DC143C	220,20,60	绯　红
Cyan	#00FFFF	0,255,255	青　色
DarkBlue	#00008B	0,0,139	深　蓝
DarkCyan	#008B8B	0,139,139	深　青
DarkGoldenRod	#B8860B	184,134,11	深金菊黄
DarkGray	#A9A9A9	169,169,169	暗　灰
DarkGrey	#A9A9A9	169,169,169	暗　灰
DarkGreen	#006400	0,100,0	深　绿
DarkKhaki	#BDB76B	189,183,107	深卡其色
DarkMagenta	#8B008B	139,0,139	深品红
DarkOliveGreen	#556B2F	85,107,47	深橄榄绿
DarkOrange	#FF8C00	255,140,0	深　橙
DarkOrchid	#9932CC	153,50,204	深洋兰紫
DarkRed	#8B0000	139,0,0	深　红
DarkSalmon	#E9967A	233,150,122	深鲑红

续表

颜 色 名 称	RGB 十六进制	RGB 十进制	中 文 名
DarkSeaGreen	#8FBC8F	143,188,143	深海藻绿
DarkSlateBlue	#483D8B	72,61,139	深岩蓝
DarkSlateGray	#2F4F4F	47,79,79	深岩灰
DarkSlateGrey	#2F4F4F	47,79,79	深岩灰
DarkTurquoise	#00CED1	0,206,209	深松石绿
DarkViolet	#9400D3	148,0,211	深 紫
DeepPink	#FF1493	255,20,147	深 粉
DeepSkyBlue	#00BFFF	0,191,255	深天蓝
DimGray	#696969	105,105,105	昏 灰
DimGrey	#696969	105,105,105	昏 灰
DodgerBlue	#1E90FF	30,144,255	湖 蓝
FireBrick	#B22222	178,34,34	火砖红
FloralWhite	#FFFAF0	255,250,240	花卉白
ForestGreen	#228B22	34,139,34	森林绿
Fuchsia	#FF00FF	255,0,255	洋 红
Gainsboro	#DCDCDC	220,220,220	庚氏灰
GhostWhite	#F8F8FF	248,248,255	幽灵白
Gold	#FFD700	255,215,0	金 色
GoldenRod	#DAA520	218,165,32	金菊黄
Gray	#808080	128,128,128	灰 色
Grey	#808080	128,128,128	灰 色
Green	#008000	0,128,0	调和绿
GreenYellow	#ADFF2F	173,255,47	黄绿色
HoneyDew	#F0FFF0	240,255,240	蜜瓜绿
HotPink	#FF69B4	255,105,180	艳 粉
IndianRed	#CD5C5C	205,92,92	印度红
Indigo	#4B0082	75,0,130	靛 蓝
Ivory	#FFFFF0	255,255,240	象牙白
Khaki	#F0E68C	240,230,140	卡其色
Lavender	#E6E6FA	230,230,250	薰衣草紫
LavenderBlush	#FFF0F5	255,240,245	薰衣草红
LawnGreen	#7CFC00	124,252,0	草坪绿
LemonChiffon	#FFFACD	255,250,205	柠檬绸黄
LightBlue	#ADD8E6	173,216,230	浅 蓝
LightCoral	#F08080	240,128,128	浅珊瑚红
LightCyan	#E0FFFF	224,255,255	浅 青
LightGoldenRodYellow	#FAFAD2	250,250,210	浅金菊黄
LightGray	#D3D3D3	211,211,211	亮 灰
LightGrey	#D3D3D3	211,211,211	亮 灰
LightGreen	#90EE90	144,238,144	浅 绿
LightPink	#FFB6C1	255,182,193	浅 粉
LightSalmon	#FFA07A	255,160,122	浅鲑红
LightSeaGreen	#20B2AA	32,178,170	浅海藻绿




续表

颜色名称	RGB 十六进制	RGB 十进制	中文名
LightSkyBlue	#87CEFA	135,206,250	浅天蓝
LightSlateGray	#778899	119,136,153	浅岩灰
LightSlateGrey	#778899	119,136,153	浅岩灰
LightSteelBlue	#B0C4DE	176,196,222	浅钢青
LightYellow	#FFFFE0	255,255,224	浅　黄
Lime	#00FF00	0,255,0	绿　色
LimeGreen	#32CD32	50,205,50	青柠绿
Linen	#FAF0E6	250,240,230	亚麻色
Magenta	#FF00FF	255,0,255	洋　红
Maroon	#800000	128,0,0	栗　色
MediumAquaMarine	#66CDAA	102,205,170	中碧绿
MediumBlue	#0000CD	0,0,205	中　蓝
MediumOrchid	#BA55D3	186,85,211	中洋兰紫
MediumPurple	#9370DB	147,112,219	中　紫
MediumSeaGreen	#3CB371	60,179,113	中海藻绿
MediumSlateBlue	#7B68EE	123,104,238	中岩蓝
MediumSpringGreen	#00FA9A	0,250,154	中嫩绿
MediumTurquoise	#48D1CC	72,209,204	中松石绿
MediumVioletRed	#C71585	199,21,133	中紫红
MidnightBlue	#191970	25,25,112	午夜蓝
MintCream	#F5FFFA	245,255,250	薄荷乳白
MistyRose	#FFE4E1	255,228,225	雾玫瑰红
Moccasin	#FFE4B5	255,228,181	鹿皮色
NavajoWhite	#FFDEAD	255,222,173	土著白
Navy	#000080	0,0,128	藏　青
OldLace	#FDF5E6	253,245,230	旧蕾丝白
Olive	#808000	128,128,0	橄榄色
OliveDrab	#6B8E23	107,142,35	橄榄绿
Orange	#FFA500	255,165,0	橙　色
OrangeRed	#FF4500	255,69,0	橘　红
Orchid	#DA70D6	218,112,214	洋兰紫
PaleGoldenRod	#EEE8AA	238,232,170	白金菊黄
PaleGreen	#98FB98	152,251,152	白绿色
PaleTurquoise	#AFEEEE	175,238,238	白松石绿
PaleVioletRed	#DB7093	219,112,147	白紫红
PapayaWhip	#FFEFD5	255,239,213	番木瓜橙
PeachPuff	#FFDAB9	255,218,185	粉扑桃色
Peru	#CD853F	205,133,63	秘鲁红
Pink	#FFC0CB	255,192,203	粉　色
Plum	#DDA0DD	221,160,221	李　紫
PowderBlue	#B0E0E6	176,224,230	粉末蓝
Purple	#800080	128,0,128	紫　色
RebeccaPurple	#663399	102,51,153	丽贝卡紫

续表

颜色名称	RGB十六进制	RGB十进制	中文名
Red	#FF0000	255,0,0	红　色
RosyBrown	#BC8F8F	188,143,143	玫瑰褐
RoyalBlue	#4169E1	65,105,225	品　蓝
SaddleBrown	#8B4513	139,69,19	鞍　褐
Salmon	#FA8072	250,128,114	鲑　红
SandyBrown	#F4A460	244,164,96	沙　褐
SeaGreen	#2E8B57	46,139,87	海藻绿
SeaShell	#FFF5EE	255,245,238	贝壳白
Sienna	#A0522D	160,82,45	土黄赭
Silver	#C0C0C0	192,192,192	银　色
SkyBlue	#87CEEB	135,206,235	天　蓝
SlateBlue	#6A5ACD	106,90,205	岩　蓝
SlateGray	#708090	112,128,144	岩　灰
SlateGrey	#708090	112,128,144	岩　灰
Snow	#FFFAFA	255,250,250	雪　白
SpringGreen	#00FF7F	0,255,127	春　绿
SteelBlue	#4682B4	70,130,180	钢　青
Tan	#D2B48C	210,180,140	日晒褐
Teal	#008080	0,128,128	鸭翅绿
Thistle	#D8BFD8	216,191,216	蓟　紫
Tomato	#FF6347	255,99,71	番茄红
Turquoise	#40E0D0	64,224,208	松石绿
Violet	#EE82EE	238,130,238	紫罗兰色
Wheat	#F5DEB3	245,222,179	麦　色
White	#FFFFFF	255,255,255	白　色
WhiteSmoke	#F5F5F5	245,245,245	烟雾白
Yellow	#FFFF00	255,255,0	黄　色
YellowGreen	#9ACD32	154,205,50	暗黄绿色

注意：RebeccaPurple是CSS Level 4中的一种新颜色，是Web Community全体成员以队友Eric去世的女儿Rebecca命名的，以此来支持他。